Angerhausen · Becker · Gerits
Schellenberger

A BASIC programozás magasiskolája a C 64-esen

Programtervezés nemcsak szakembereknek

DATA BECKER – NOVOTRADE

Angerhausen · Becker · Gerits
Schellenberger

A BASIC programozás magasiskolája a C 64-esen

Programtervezés nemcsak szakembereknek

DATA BECKER – NOVOTRADE

A mű eredeti címe: 64 für Profis (1984)

Fordította: DOBOSNÉ HARTYÁNYI MÁRIA
Lektorálta: DR. OBÁDOVICS J. GYULA és DR. LENGYEL JÓZSEF
A programokat lektorálta: SZERENCSI RÓBERT

A kiadásért felel: RÉNYI GÁBOR, a NOVOTRADE RT. igazgatója
Kiadványmenedzser: BÉKÉS TAMÁS
Sorozatszerkesztő és a kötetet szerkesztette: ROCHLITZ ANDRÁS
Műszaki szerkesztő: DÉVÉNYI ERIKA

Szedte: az Alföldi Nyomda, Debrecen
Készült a Nyírségi Nyomdában, Nyíregyházán. 11,5 (A/5 ív)
Budapest, 1985.
ISBN 963 02 4036 X

© Hungarian translation Dobosné Hartyányi Mária

Copyright © 1984 DATA BECKER GmbH – Merowingerstr. 30. 4000 Düsseldorf

Minden jog fenntartva. A DATA BECKER cég írásbeli hozzájárulása nélkül tilos a jelen könyvet vagy annak részeit bármilyen eljárással (nyomtatás, fotókópia vagy egyéb technika), elektronikus rendszerek felhasználásával másolni, sokszorosítani, terjeszteni.

FONTOS TUDNIVALÓ

A jelen könyv keretén belül ismertetett kapcsolások, eljárások és programok nem tekintendők szabadalmi oltalom alá eső ipari termékeknek. Ezek elsősorban amatőr és oktatási célokat szolgálnak. A szerzők rendkívül nagy gondot fordítottak a kapcsolások, műszaki adatok és programok helyességére, a részletek kidolgozása során többszöri ellenőrzést végeztek. Mindez azonban nem zárja ki az esetleges hibalehetőségeket.
Az előforduló hibákért és az ebből adódó következményekért a DATA BECKER cég sem szavatosságot, sem jogi felelősséget nem vállal. Az esetlegesen előforduló hibák közlését a szerzők hálásan fogadják.

ELŐSZÓ

Kötetünk, melyet az Olvasó most kezében tart, már a hetedik a DATA BECKER könyvek kedvelt sorozatában. Az a nap, amelyen megjelent, különösen jelentős a DATA BECKER könyvek életében.
A megjelenését megelőző 10 hónap alatt ugyanis több mint 100 000 DATA BECKER könyvet adtunk el.
Szerzőink ezen könyv megírásakor is arra törekedtek, hogy hasznos tanácsokkal és gyakorlati példákkal segítsék a C 64-es gép sokoldalú, optimális alkalmazását.
A könyv témája: felhasználói programok készítése BASIC nyelven, magas színvonalú programozástechnikával.
Szó esik majd programtervezésről, menűvezérlésről, a képernyőmaszkok célszerű felépítéséről, a programok paraméterezéséről, adatkezeléséről, ill. a programok dokumentálásának módszeréről, vagyis mindenről, ami egy igazán jó program elkészítéséhez szükséges.
Különösen büszkék vagyunk a Klaus Gerits által kifejlesztett új QUISAM file elérési módszerére.
A címlapon szereplő szerzőkön kívül – akik más DATA BECKER könyvekből már ismerősek lehetnek az Olvasó számára – Englisch, Froitzheim, Kausmann és Retzlaff urak segítettek a könyv elkészítésében. A rajzok Jordan asszonytól származnak. Mindannyiuknak köszönetet mondunk munkájukért.
A könyv olvasásához jó szórakozást kívánnak

a Szerzők.

TARTALOMJEGYZÉK

ELŐSZÓ 5

1. FEJEZET
Mielőtt elkezdenénk programot írni 11

1.1 Ettől kímélje meg magát! 11
1.2 Az ötlet 12
1.3 A programtervezés 16
1.3.1 A folyamatábra 16
1.4 A programírás közvetlen előkészületei 28
1.4.1 A képernyőmaszkok megtervezése 28
1.4.2 A file-ok megtervezése 30
1.4.3 Algoritmusok, számítások 34
1.4.4 A nyomtatás 34
1.4.5 A változólista – ötletek a változók megválasztására 35
1.4.6 A program dokumentációja 37
1.4.7 Gondolatok a program szerkezeti felépítéséről 37
1.5 A strukturált programozás alapelemei egy példán keresztül: a Commodore 64-es, mint zsebszámológép 39
1.5.1 Az előkészítés 39
1.5.2 A programelemzés 39
1.5.3 A változólista 40
1.5.4 A program felépítése 41
1.6 A struktogram 45
1.7 A folyamatábra és a struktúradiagram összehasonlítása 64
1.8 A strukturált BASIC program 65

2. FEJEZET
A program részei 69

2.1 Az adatbevitel 70
2.1.1 A képernyőmaszk 70
2.1.2 A beviteli mezők 71
2.1.3 A kurzor pozícionálása 75
2.1.4 Beolvasás a billentyűzetről 76
2.2 Az adatfeldolgozás 80
2.2.1 A változók 80
2.2.2 A számítási pontosság 81
2.2.3 A rendezés 82
2.3 Az adatok kiírása 87
2.3.1 Az adatok kiírása lemezre 87
2.3.1.1 A SEQ 88

2.3.1.2 A REL 89
2.3.1.3 Az ISAM 93
2.3.1.4 A QUISAM 94
2.3.1.4.1 A HASH kód 95
2.3.1.4.2 A file mérete 96
2.3.1.4.3 Egy rekord törlése a QUISAM file-ból 103
2.3.1.4.4 Egy rekord felírása a file-ba 108
2.3.1.4.5 Egy rekord keresése a QUISAM file-ban 111
2.3.1.4.6 Mire alkalmas a QUISAM file? 113
2.3.2 A nyomtatás 113
2.3.2.1 Nyomtató = Nyomtató? 114
2.3.2.2 A tabulátor 115
2.3.2.3 A tizedespontok 116
2.3.2.4 A nyomtatáskép formátuma 116

3. FEJEZET
Ötletek a magasszintű programozástechnikához 120

3.1 A programok paraméterezése 120
3.2 Menüvezérlés és overlay-technika 123
3.3 A funkcióbillentyűk szerepe a fejlett programtechnikában 126
3.4 Teszteljük programjainkat! 128
3.5 A kezelési utasítás a programozó névjegye 130
3.6 Felülről lefelé vagy lentről felfelé? 132
3.7 A logikai típusú változók – nyolc lehetőség byte-onként 134
3.8 A dátum, mint különleges adattípus 136
3.9 Az adatok tömörített tárolása – egy rekordban több mint 255 karakter 137

4. FEJEZET
Programpéldák 139

4.1 Bevezetés 139
4.2 Paraméterezett raktárnyilvántartás 140
4.3 Egy általános listázó 153
4.4 Egy egyszerű szövegfeldolgozó program 156
4.5 A QUISAM alkalmazása – irodalomjegyzékek nyilvántartása 161
4.6 Kényelmes címkezelés 167

5. FEJEZET
A programozási segédletek felhasználása 171
5.1 A MASTER 64 171
5.2 A STRUKTO 64 178
5.3 A grafika programozási elvei a STRUKTO 64-ben 182

BEVEZETÉS

Az igazi szakemberek másként programoznak?
Igen is és nem is. A szakemberek mindenekelőtt más célra programoznak, mint műkedvelő társaik. Rendszerint olyan programokat készítenek, amelyeket nem saját maguk, hanem más emberek használnak. Ebből adódóan a programok tervezésének és kivitelezésének szükségszerűen más-más módszert kell követniük. Gondoljon vissza az Olvasó azokra az időkre, amikor programozni tanult. Kezdetben már az is nagy örömet jelentett, ha egy egyszerű utasításnak a gép engedelmeskedett. A képernyő minden reakciója, minden apró eredmény büszkeséggel töltötte el Önt. Az új utasítások és programozási trükkök ismeretének gyarapodásával a programok is egyre növekednek. Biztosan előfordult már az Olvasóval is, hogy egy idő után saját maga sem ismeri ki magát a program útvesztőiben és a hibakeresés ilyenkor kínszenvedéssé válik. Képzelje csak el, ha ezzel a programmal valaki másnak kellene dolgoznia!
A hivatásos programozó természetesen nem így dolgozik. Számára a programozás nem öncél, és nem hobby (ami persze nem zárja ki azt, hogy kedvét lelje benne), hanem egy meghatározott munka elvégzése. A programozási feladatot a lehető legjobban és leggyorsabban kell végrehajtania, ezért minden programozó fontosnak tartja az alábbiakat:

Az eredményes programkészítés módszere.

Az idő pénz. A hivatásos programozó célja, hogy a lehető leggyorsabban, a lehető legkisebb ráfordítással futtatható programot készítsen. Számára egy új program nem ötletek kusza kivitelezése, hanem gondos tervezés és előkészítés eredménye. A gyakorlott programozó általában „elemekből" építkezik.
Az egyedi programozási feladatok mindig tartalmaznak közös elemeket, azonos modulokat (maszkfelépítés, file-kezelés, adatkezelés, számítási képletek stb.), ezekből az elemekből gyorsan összeállíthatja az új feladat vázlatát, és így a további munka során figyelmét a kódolásról a szervezésre fordíthatja. A program gondos tervezése megvédi készítőjét a programozási zsákutcáktól és kerülőutaktól. Aki programját előzőleg jól megtervezi, az később már nincs ráutalva pl. a hobby-programozóknál oly kedvelt átsorszámozásra. A hivatásos programozót nem annyira a zseniális programozási trükkök (itt gyakran fölébe kerekednek az elkötelezett hobby-programozók), mint inkább a céltudatos munka jellemzi.

A programírás közben nem szabad megfeledkezni a program jövendő használóiról!
Aki nem saját maga számára készít programokat, annak nagyon gondosan kell eljárnia. Az ilyen programoknak könnyen kezelhetőknek kell lenniük és sosem szabad olyan esetnek előfordulnia, hogy a felhasználó a képernyőn látható kezelési útmutató hiányában tanácstalanul lapozgatni kényszerüljön a programról szóló kézikönyvet. Erre egy példa:
Gondoljon egy bizonyos útszakaszra, amit kocsival gyakran megtesz. Önnek ez nem jelent problémát, mert annyira ismeri az utat, hogy szinte bekötött szemmel is bármikor megteszi. Most gondoskodjék arról, hogy egy helyismerettel nem rendelkező személy is könnyen megtehesse ezt az útszakaszt anélkül, hogy eltévedne. Két segédeszköz áll az Ön rendelkezésére. Az egyik a jelzőtáblák használata, ami a képernyőn megjelenő tájékoztatásnak felel meg. (Ez utóbbira szerencsére nagyobb befolyásunk van, mint a közúti közlekedés jelzőtábláira). A másik egy írott útmutató, ami a programleírásnak, illetve a használói kézikönyvnek felel meg. Ha az Olvasó komolyan végiggondolja a feladatot, biztosan megérti, hogy mi a különbség a saját, házi használatra készített programok és az idegen felhasználók számára

írt programok között! Aki professzionális igényeknek megfelelő programokat akar készíteni, annak gondolnia kell a „helyismerettel nem rendelkezők"-re is!

A programok legyenek könnyen módosíthatóak.

A programok bizonyos környezeti feltételek figyelembevételével készülnek és ezek a környezeti feltételek könnyen megváltozhatnak. Ezért nagyon fontos, hogy a programokat is könnyen a megváltozott körülményekhez lehessen igazítani. A megrendelő új, előre nem látható kívánságai vagy a megváltozott peremfeltételek, például egy új értéktöbblet-adókulcs, egy új kereseti adóképlet vagy egy új cégjelzés egy jól megírt program számára nem jelenthetnek problémát. Jaj azonban azoknak, akiknek ilyen esetben egy 20 kbyte-os „spagetti-kódon" kell átrágniuk magukat és míg minden egyes hiba megszüntetése új hibákat hoz felszínre, vágyakozva gondolnak a kurzorra, pedig az ilyenkor csak a vakvezető kutya szerepét tölthetné be.

Ennek a könyvnek nem az a célja, hogy az Olvasót szuperprogramozóvá képezze át, aki egy új VISICALC-ot, vagy olyan nagy csodát varázsol elő a ruhaujjából, mint a közismert WORDSTAR. Mi csupán azt szeretnénk az Olvasónak megmutatni, hogy miként érhet el jobb eredményt kevesebb ráfordítással és hogyan készíthet olyan programokat, amelyekben nemcsak önmaga, hanem más emberek is örömüket lelhetik. A Commodore 64-es ehhez kitűnő feltételeket kínál. 64 kbyte-os memóriájával többet nyújt, mint amit pár évvel ezelőtt az átlagos adattechnika korszerű berendezései nyújtani tudtak.
Elméleti fejtegetésekkel ebben a könyvben a lehető legkevesebbet szeretnénk untatni az Olvasót, még ha néha elkerülhetetlen is. Elmélet helyett inkább megvalósítható tanácsokat és kipróbált recepteket igyekszünk nyújtani, mindig szem előtt tartva az alábbiakat:

A céltudatosság és a tervszerűség a programozásban is, sőt a programozásban különösen megtérül!

1. FEJEZET
MIELŐTT ELKEZDENÉNK PROGRAMOT ÍRNI...

1.1 Ettől kímélje meg magát!

A legtöbb kedvtelésből programozó (alkalmanként a hivatásos programozók is) programjaikat minden előkészület nélkül közvetlenül begépelik. Azaz: az ötlettől a kész programig mindent kizárólag a gépen végeznek. Egyszerű programoknál ez a módszer még elfogadható, nagyobb és igényesebb feladatoknál azonban már nem.
Bizonyára az Olvasóval is előfordult, hogy hirtelen támadt egy ötlete, odaült a Commodore 64-es elé, bekapcsolta a gépet és azonnal nekilátott a programozásnak. Kezdetben minden jól ment. Már néhány programsor után – melyek még jól áttekinthetőek voltak – működőképes volt a program és a futás eredménye látható volt a képernyőn. Programozás közben azonban az eredeti ötlet folyamatosan módosult, a program nőttön-nőtt. Az ötlet végleges formáját követve, a programot itt is, ott is javítgatni kellett, és az egyre kuszábbá, áttekinthetetlenebbé vált.
Mentőövként szolgálnak ilyenkor a fuldoklónak a programozási segédeszközök: a FIND, a DUMP, a RENUMBER stb. Általában azonban ezekkel az eszközökkel is csak a jelenségeket lehet kezelni és nem azok okát. A hibák halmozódása, a program áttekinthetetlensége egy idő után legtöbbször kedvét szegi a vállalkozónak, a program bekerül a „soha el nem készülő" programok gyűjteményébe.
Mi történt valójában? Jó öreg favágó módjára „bevágta" Commodore 64-esébe a kódokat, de nem programozott. A programozás ugyanis sokkal több ennél. Azt, amit Ön csinált, kódolásnak nevezik, ami azt jelenti, hogy a feladat elemi lépéseit lefordítottuk a gép által érthető nyelvre, pl. BASIC-re. A programozáshoz azonban még sok minden más is tartozik, nemcsak a kódolás.
A munka a programötlet gondos kidolgozásával kezdődik. Szaknyelven ezt program-, illetve rendszerelemzésnek nevezik. Az elektronikus adatfeldolgozásban erre a feladatra jól fizetett és megfelelően kvalifikált rendszerszervezőket foglalkoztatnak. A rendszerszervezők az új alkalmazások programterveit gondosan elemzik és előkészítik úgy, hogy az könnyen kódolható legyen.
Mivel az Olvasó bizonyára nem tarthat magának saját rendszerszervezőt, célszerű megtanulnia hogy hogyan történik a gondos programanalízis.
Az alábbi fejezetben azt szeretnénk bebizonyítani, hogy zseniális programozási ötletek nélkül, pusztán a gondos előkészítő munkával, a feladat lépésenkénti elemzésével is lehet – az üzemzavart elkerülve – gyorsan és hatékonyan programozni.

1.2 Az ötlet

Tagadhatatlan, hogy a számítástechnikában – még sokkal inkább, mint az élet más területén – kincset ér egy-egy jó programötlet. Bizonyára Ön is, kedves Olvasó, sokat beszélhetne erről. A legtöbb programozó – ahogy támadt egy jó ötlete – azonnal odaül a géphez és megpróbálja programmá alakítani az ötletet. A tapasztalat szerint ezzel a módszerrel hosszú, akadályokkal teli rögös út vezet a megoldáshoz. Pontosan ezeket a programtechnikai akadályokat szeretnénk kiküszöbölni azzal, hogy a feladatot előkészítjük, mielőtt a gépet egyáltalán bekapcsoljuk. A hibák kiküszöbölésének alapvető feltétele az adott probléma részletes kidolgozása elejétől végéig.
Minden program elején az ötlet áll. Mivel az ember az ötlet születésének pillanatában tudja a legjobban, hogyan kell kinéznie a jövőbeni programnak, ezért az a legcélszerűbb, ha az ötletet rövid címszavakban körülírjuk.
A későbbiekben (az íróasztalnál) pontosan meg kell határozni, hogy mit kell tudnia a programnak. Ezután össze kell gyűjteni az összes kritikus pontot és a programhoz szükséges egyéb részleteket. Érdemes utánanézni, hogy készítettünk-e már olyan programokat, amelyekben hasonló problémák merültek fel. Biztosan találunk néhány olyan rutint, amelyet változtatás nélkül beépíthetünk az új programba.
Ha az ötletet pontosan megfogalmaztuk, megtettük az első lépést a program előkészítéséhez. Ezt az első lépést szaknyelven *feladat-analízis*nek nevezik. Tekintsük ezt a problémaelemzést egyfajta anyaggyűjtésnek. A következőkben kis példák alapján bemutatjuk, hogy mit kell tartalmaznia a feladat elemzésének. Az elemzés során még nem kell jelentőséget tulajdonítanunk annak, hogy hogyan fogalmazzuk majd meg a feladatot a gép számára érthető nyelven.

PÉLDA:
Egy címnyilvántartási rendszer elemzése

Szeretnénk egy programot írni, amellyel nyilvántarthatjuk a számunkra fontos címeket. Az első kérdés, amelyre a probléma felvetésekor azonnal válaszolnunk kell:
Mi lesz a program feladata?
A program tegye lehetővé a címek bevitelét, tárolását és visszaolvasását. Másrészt adjon lehetőséget arra, hogy egyes adatokat megkereshessünk és a képernyőre írathassunk.
Miután meghatároztuk a program célját, bontsuk részekre a feladatot. Minden részt külön elemzünk és a szükséges kritériumok figyelembevételével gondosan megtervezünk. Címadminisztrációnk a következő részekre tagolódik:

- adatbevitel
- file-kezelés
- adatkeresés

Adatbevitel

Az adatbevitelhez át kell gondolnunk, hogy tulajdonképpen milyen adatokat kell tárolni. El kell döntenünk, hogy egy rekord milyen adatmezőket tartalmazzon és meg kell határoznunk, hogy egy-egy adatmező maximum hány karakterből állhat.
Az adatmezők száma és hosszúsága alapján ki tudjuk számítani, hogy hány rekord fér el maximálisan a Commodore 64-es operatív tárában.
Végezzük el ezt a számítást a címnyilvántartásra vonatkozóan.

Egy cím az alábbi adatmezőket tartalmazza:

Vezetéknév	=	15 karakter
Keresztnév	=	15 karakter
Utca	=	15 karakter
Postai irányítószám	=	4 karakter
Telefon	=	13 karakter
Megjegyzés	=	25 karakter
Rekordhossz	=	87 karakter

Mint tudjuk, a Commodore 64-esen 38 911 byte-ot használhatunk szabadon az adatok és a program tárolására.
Mivel azt nem tudjuk pontosan előre megmondani, hogy mekkora területet foglal majd el a program, azt sem tudjuk kiszámítani, hogy hány címrekord fér majd el a tárban. A számítás menetét azonban enélkül is meggondolhatjuk. A program beírása után a FRE (X) függvénnyel lekérdezhetjük a szabad byte-ok számát. Mivel a rekord minden karaktere egy byte-ot foglal le, a maximális rekordszám meghatározásához nem kell mást tennünk, mint elosztani a kapott értéket a rekord hosszával. Tegyük fel, hogy a szabad tárterület 29 568 byte. Ekkor a tárolható rekordok maximális száma:

29568 : 87 = 339,86

Ebben az esetben tehát a tárban 339 címet kezelhetnénk. Állapodjunk meg abban, hogy a programmal 200 rekordot, azaz 200 címet kezelünk. A fenti számítással meggyőződhetünk arról, hogy ennyi el is fér a tárban. Ezzel tisztáztuk az adatbevitelre vonatkozó problémákat, továbbléphetünk a file-kezelés felé.

File-kezelés

Mivel nem akarjuk az adatokat minden alkalommal újra begépelni, meg kell oldanunk a tárolás kérdését. Ahhoz, hogy az adatokat külső adathordozón tárolni tudjuk, szükségünk lesz egy file-ra. Azonnal felmerül a következő kérdés: milyen szerkezetű file-t használjunk, soros, vagy közvetlen elérésű file-t?
A Commodore 64-esen minden bizonnyal egyszerűbb a soros file-szervezés (sőt kazettás egységen csak soros file-t tudunk kezelni). Ha az Olvasó rendelkezik VC–1541-es lemezegységgel, akkor három lehetséges file-típus közül választhat:

soros file
közvetlen elérésű file
relatív file

Mi soros file-típussal fogunk dolgozni, azért, hogy a programot azok is használhassák, akik csak kazettás egységgel rendelkeznek. Azok számára, akik rendelkeznek VC–1541-es lemezegységgel, útmutatóul szolgálhat, a már korábban megjelent *„A VC–1541-es lemezegység programozása"* c. könyv.
A kérdést tehát eldöntöttük: a programot soros file-kezelésre tervezzük!
Abban az esetben, ha a programmal egyszerre több címnyilvántartást szeretnénk kezelni (pl. cégcímek, szállodacímek, ismerősök címei stb.), akkor a programot fel kell készíteni arra, hogy egyszerre több file-t is tudjon kezelni. A programozás során tehát ügyelnünk kell arra, hogy a file neve változtatható legyen!

Ez a tény különösen akkor fontos, ha lemezegységen tároljuk az adatokat. A lemezegységen ugyanis minden adathalmaznak külön nevet kell viselnie, a DOS ugyanis a név alapján azonosítja az adathalmazokat. A kazettás egységen egy névvel több file-t is tárolhatunk. Ez persze nem célszerű, mert egy idő után a szalag áttekinthetetlenné válik.
Sokkal jobb, ha az egyes file-okat külön névvel látjuk el, úgy, hogy a file neve emlékeztessen a tartalmára.
Attól függően, hogy a programot mennyire „összkomfortosnak" tervezzük, a file-kezelésre vonatkozóan még felmerülhetnek a következő kérdések:

- bővíthető legyen-e a file?
- kell-e módosítani időnként a tárolt adatokat?
  (ha pl. raktárkészleteket tárolunk)
- tudjon-e a program file-t törölni?
- tudjon-e a file-okról másolatot készíteni a program?

Az első kérdésre válaszoljunk igennel. Írjuk meg úgy a programot, hogy a címnyilvántartás bővíthető legyen!
A bővíthetőség nagyon fontos követelmény egy címnyilvántartás esetében. Nem sok értelme lenne ugyanis a nyilvántartásunknak, ha az állományt nem tudnánk bővíteni, kiegészíteni az új ismerősök címével.
A mágnesszalagon tárolt file bővítésének lépései:

1. a file beolvasása
2. a file bővítése adatbevitellel
3. a file újratárolása

A módosított, azaz bővített file-t vissza kell írnunk a régi adathalmaz helyére. A programot tehát úgy kell megírnunk, hogy képes legyen felülírni a fölöslegessé vált file-t!
Miután a file-típust, a file-kezelés módját meghatároztuk, vizsgáljuk meg, hogyan alakul a file belső szerkezete, azaz a file-ban milyen információt, milyen sorrendben tároljunk:

1. file-név
2. a rekordok száma
3. a rekordok

Az 1. és 2. pontra elsősorban akkor lesz szükségünk, amikor a file-t módosítanunk kell. A rekordok számának tárolása megkönnyíti pl. a ciklusszervezést a kereső rutinban.

Most, hogy a file-ra vonatkozó összes kérdésben döntöttünk, térjünk át a harmadik rész elemzésére.

Adatkeresés

Először is tisztában kell lennünk azzal, hogy hogyan és milyen szempontok szerint keressünk az adathalmazban. Elegendő-e egyetlen keresőrutint írni, amely egy keresési szempont (a mi példánkban a név, az utónév, a postai irányítószám, stb. lehet) beolvasása után megkeresi a megfelelő rekordot vagy célszerű beépíteni a programba egy további keresőrendszert?
Képzeljük el, hogy a programmal egyik ismerősünk címét szeretnénk kiíratni, de nem tudjuk

pontosan, hogy hogyan kell írni az illető nevét. Előfordulhat, hogy a program újra és újra hibajelzést ad, mert az általunk begépelt és a tárolt név nem egyezik meg betűről betűre (hasonlósági összehasonlítás nem történik). Az ilyen bosszantó esetek elkerülésére célszerű a programba egy másik keresési módot is beépíteni:

Tudjon a program kulcs szerint listázni!

Döntsük el, hogy melyik adatmező legyen a kulcs, amit a teljes rekord helyett a képernyőre ki akarunk íratni. A mi példánkban legyen a kulcs a vezetéknév. A kulcsok segítségével bármikor ellenőrizhetjük, hogy a keresett név tényleg létezik-e. Az ellenőrzés úgy történik, hogy a képernyőre kiíratjuk az összes vezetéknevet, és megnézzük, hogy közöttük van-e az általunk keresett név.
A programban lennie kell egy olyan keresőrutinnak, amely a kívánt keresési szempont beolvasása után a megfelelő rekordot automatikusan kiírja. A mi programunkban a következő szempontok alapján lehet keresni:

1. Vezetéknév
2. Keresztnév
3. Utca
4. Város
5. Telefon
6. Postai irányítószám

Ezzel be is fejezhetjük a feladat elemzését. Kérjük, hogy a programtervezés során kövesse az általunk javasolt elemzési lépcsőket. Ezzel máris megtettük az első lépést a professzionális programok készítéséhez vezető úton.

1.3 A programtervezés

A feladat elemzése során összegyűjtött anyag a programtervezés alapjául szolgálhat, ugyanis minden olyan információt tartalmaz, ami a program lényegét érinti.
Az 1.2 alfejezetben minden kérdésre közösen választ adtunk, amely a címnyilvántartási program megírása közben felmerülhet. Arról is esett már szó, hogy ha a feladat túlságosan bonyolult és áttekinthetetlen, célszerű részekre bontani.
Ilyenkor készítsünk egy vázlatot a szükséges programokról és szubrutinokról, anélkül, hogy kitérnénk a programozástechnikai részletekre. A tervezés egyik leghatásosabb eszköze a folyamatábra.
A következő alfejezetben az Olvasó megismerkedhet a folyamatábra készítésének módszerével, majd az 1.5 alfejezetben egy másik, grafikus tervezési módot, a Nassi–Schneidermann-féle struktúradiagramot mutatjuk be.

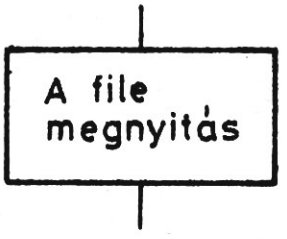
1.3.1 A folyamatábra

A folyamatábrával főként az elektronikus adatfeldolgozásban járatlan személyek számára nagyon szemléletessé és áttekinthetővé tehető egy feladat szerkezete. A folyamatábrák egyszerű és világos felépítése miatt ez az ábrázolási mód terjedt el a legszélesebb körben. Minden szakfolyóiratban, szakkönyvben találunk folyamatábrákat, amelyek a kínált programokat dokumentálják. A folyamatábrák ésszerű alkalmazásának előfeltétele azoknak a jeleknek és szimbólumoknak az ismerete, amelyeket ennél az ábrázolási módnál használnak. Az alábbiakban a DIN 66001 szabvány szerinti szimbólumokat ismertetjük.

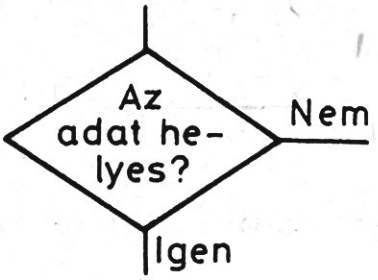
Alapelemek:

Mindössze néhány alapjelet használunk, amelyek ismeretében bárki el tud készíteni kisebb és egyszerű folyamatábrát. A folyamatábra elején, illetve végén áll az alábbi szimbólum, amellyel a program kezdetét és végét (végeit) jelöljük.

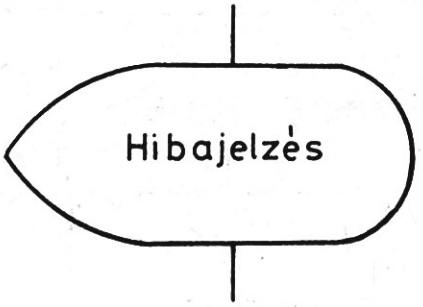
A következő szimbólum egy téglalap, amelybe beírjuk azokat az utasításokat, amelyeket a gépnek végre kell hajtania. A téglalapba írt szöveg utal az elvégzendő tevékenységre:



A következő szimbólum egy rombusz, amelybe általában egy eldöntendő kérdés kerül. A kérdésre adott válasz (pl. igen/nem) szerint a programnak más-más tevékenységet kell végrehajtania, azaz ezen a ponton a program elágazik.
Az elágazás szempontjából lényegtelen, hogy a kérdésre a választ maga a gép, vagy a gépkezelő adja meg.



Az utolsó szimbólum valamilyen kijelzésre utal, ami lehet pl. a képernyőn megjelenő szöveg, egy kibocsátott hang stb.

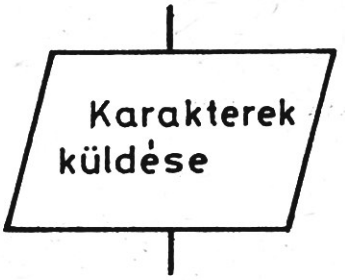


Végül érdemes néhány szót ejteni a szimbólumok szabványosításáról.
Az itt ismertetett szimbólumok a DIN 66001 (német) szabvány szerintiek. A tapasztalat azonban az, hogy szinte minden programozó saját stílust dolgozott ki a folyamatábrák készítésére az igényeinek megfelelően. Ez még önmagában nem lenne baj. De ha egy és ugyanazon a terven több programozó dolgozik, elengedhetetlen a szabványok alkalmazása (képzeljük el mi történne akkor, ha több különböző stílusú jelkészletet kellene összeilleszteni egy kész programmá?)

Egyéb szimbólumok:

A következőkben bemutatunk még néhány olyan szimbólumot, amelyek a folyamatábrákon a leggyakrabban fordulnak elő. A felsorolás a teljesség igénye nélkül készült, csak a lényeges jelekre és szimbólumokra térünk ki.

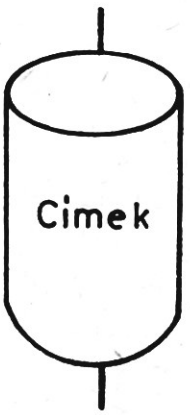
Beolvasás / Kiírás (INPUT/OUTPUT):



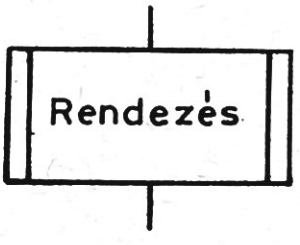
Kézi adatbevitel (MANUAL INPUT):

Listázás (DOCUMENT):

Tárolás lemezen (ONLINE STORAGE):



Szubrutin hivás (CALL SUBPROGRAM)

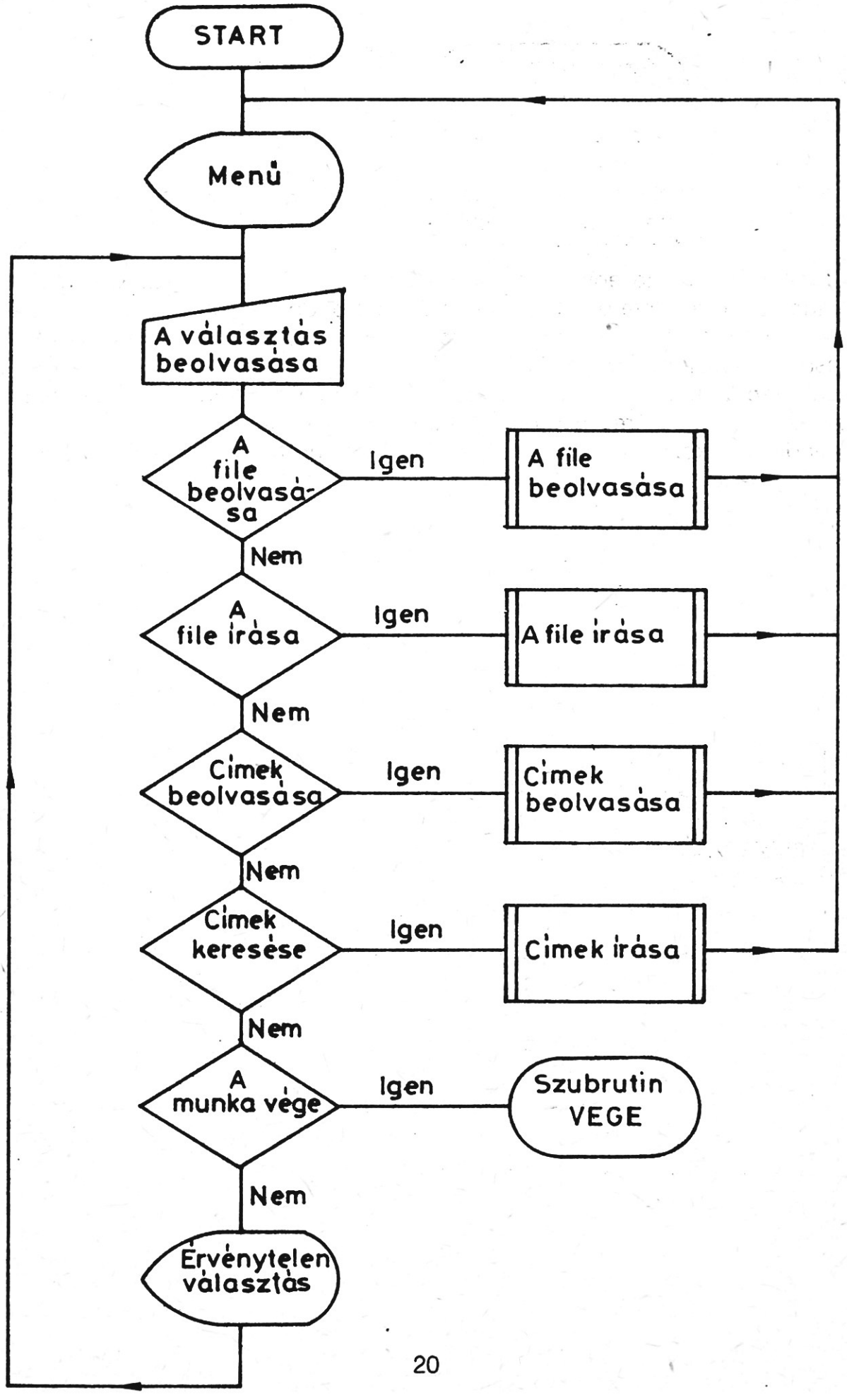


A címnyilvántartás példáján röviden bemutatjuk a folyamatábra készítésének módját. Az elemzés után már csak azt kell eldöntenünk, hogy a teljes feladat mely részeiből készítsünk szubrutint.

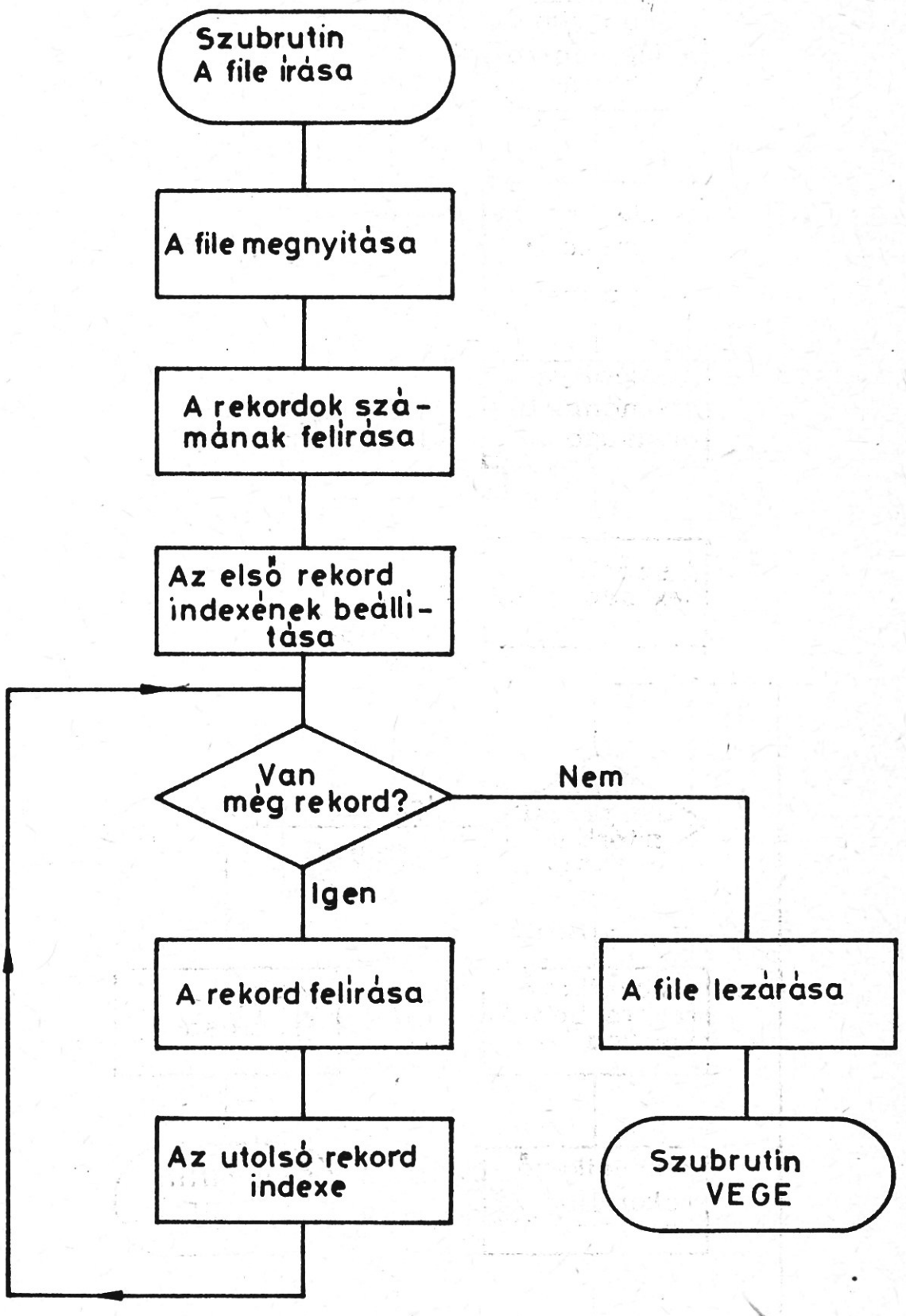
Főprogram	:	Menü és az adatbevitel ellenőrzése
1. szubrutin	:	Adatok tárolása
2. szubrutin	:	Adatok beolvasása
3. szubrutin	:	Címek beolvasása klaviatúráról
4. szubrutin	:	Címek keresése
4.1 szubrutin	:	Címlista kiírása a kulcs szerint
4.2 szubrutin	:	Címek keresése szempontok szerint

Nem lenne ésszerű, sőt néha egyenesen lehetetlen olyan folyamatábrát készíteni, amelyen a feladat minden apró részlete precízen ki van dolgozva. Sokkal jobb, ha először egy durva, vázlatos folyamattervet készítünk, ami elsősorban a főprogram és az egyes szubrutinok közötti kapcsolatrendszert tükrözi.
A durva vázlat alapján könnyebb az egyes részeket felnagyítva ábrázolni most már a kívánt precizitással, azaz az ún. alárendelt folyamatábrákat elkészíteni.
Ezek után minden eszköz rendelkezésünkre áll ahhoz, hogy a címnyilvántartási rendszert grafikusan ábrázoljuk.

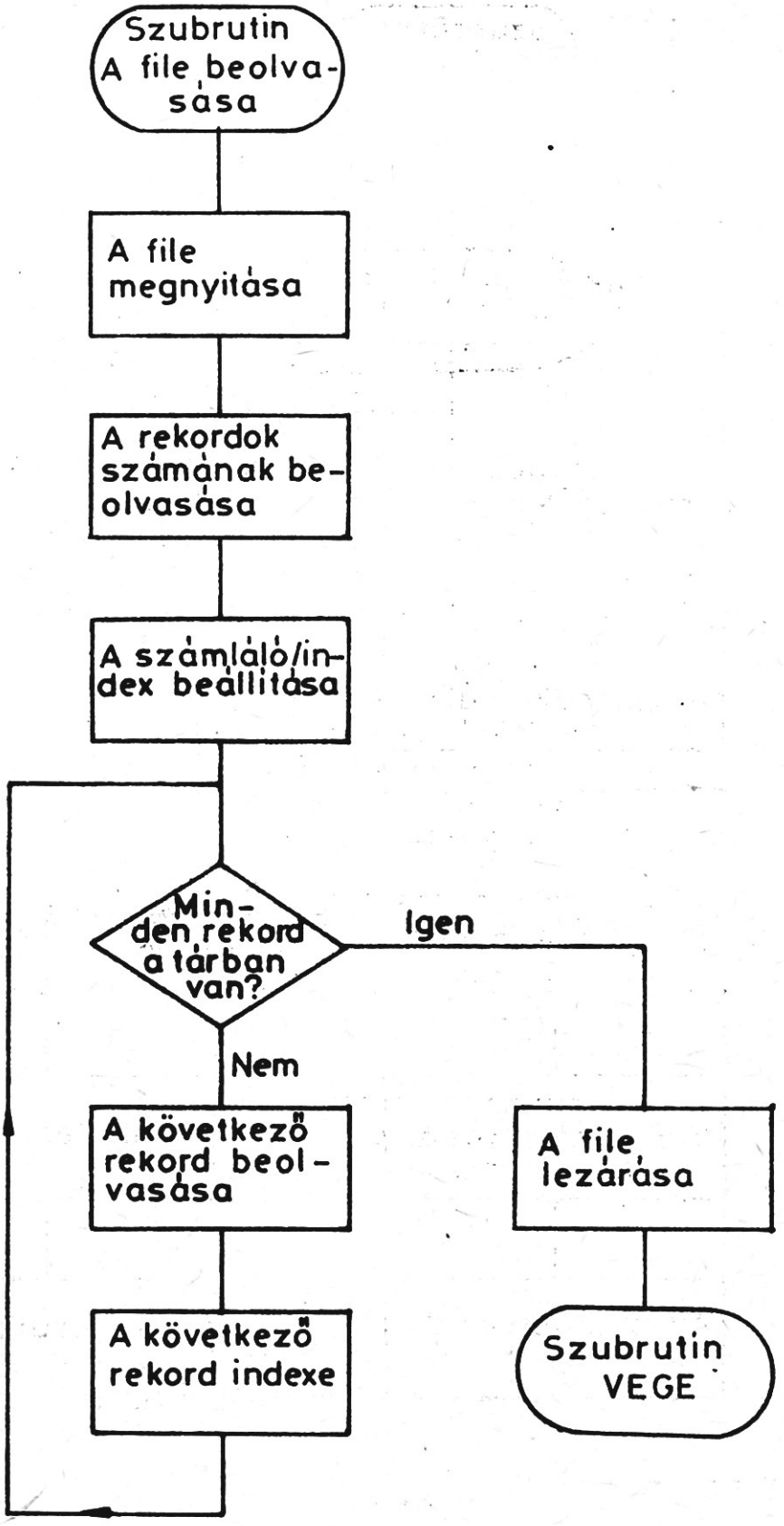
Főprogram: Menű
Szubrutin hívás választás szerint



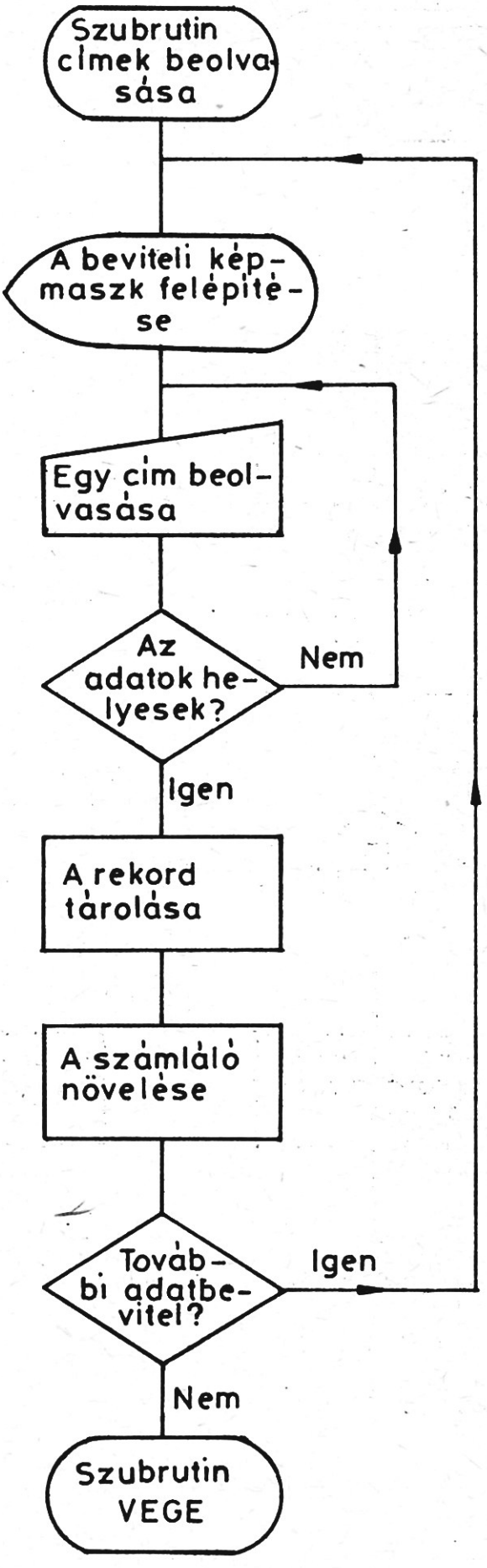
1. SZUBRUTIN: az adatok tárolása

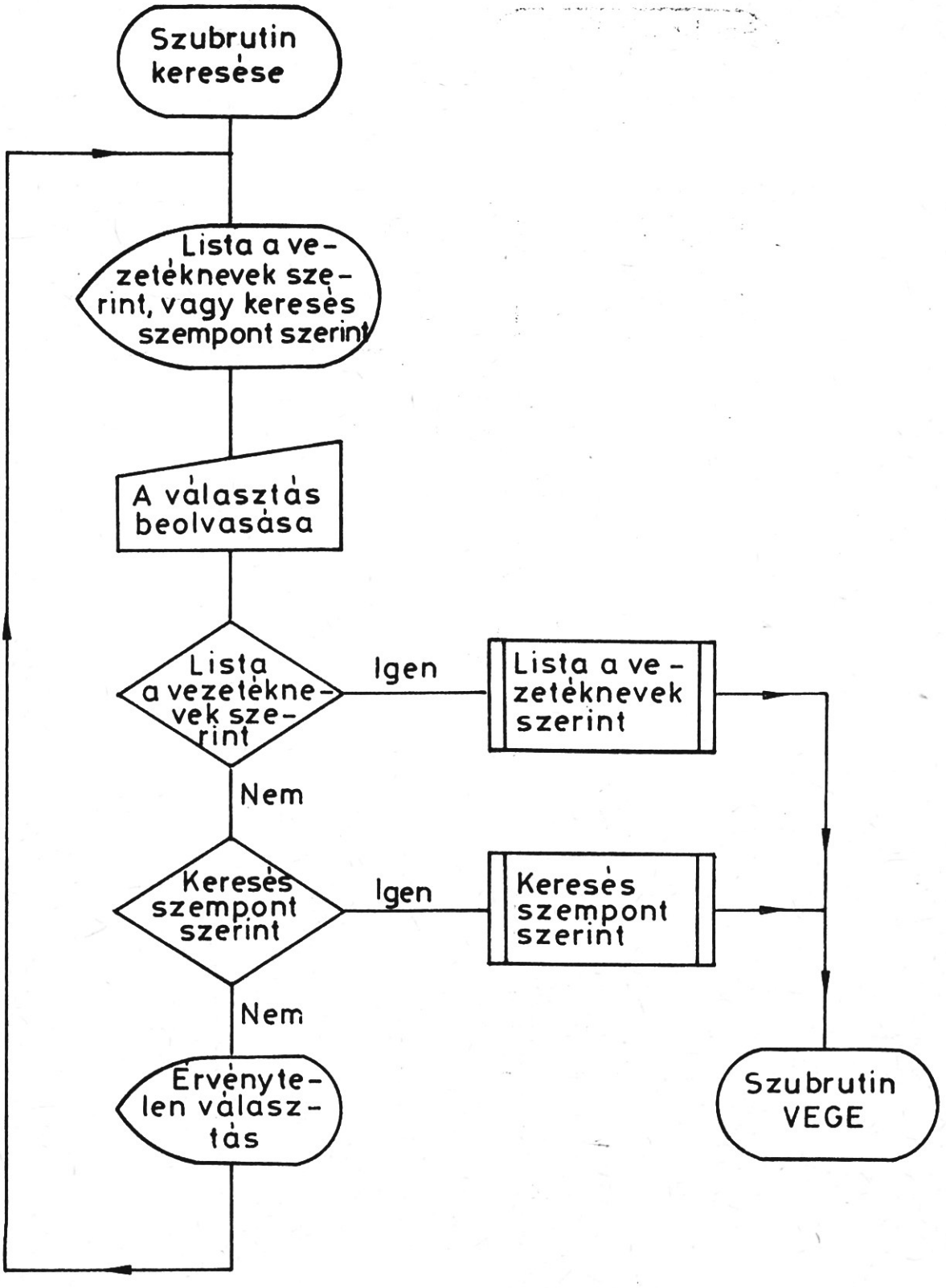


2. SZUBRUTIN: a file beolvasása

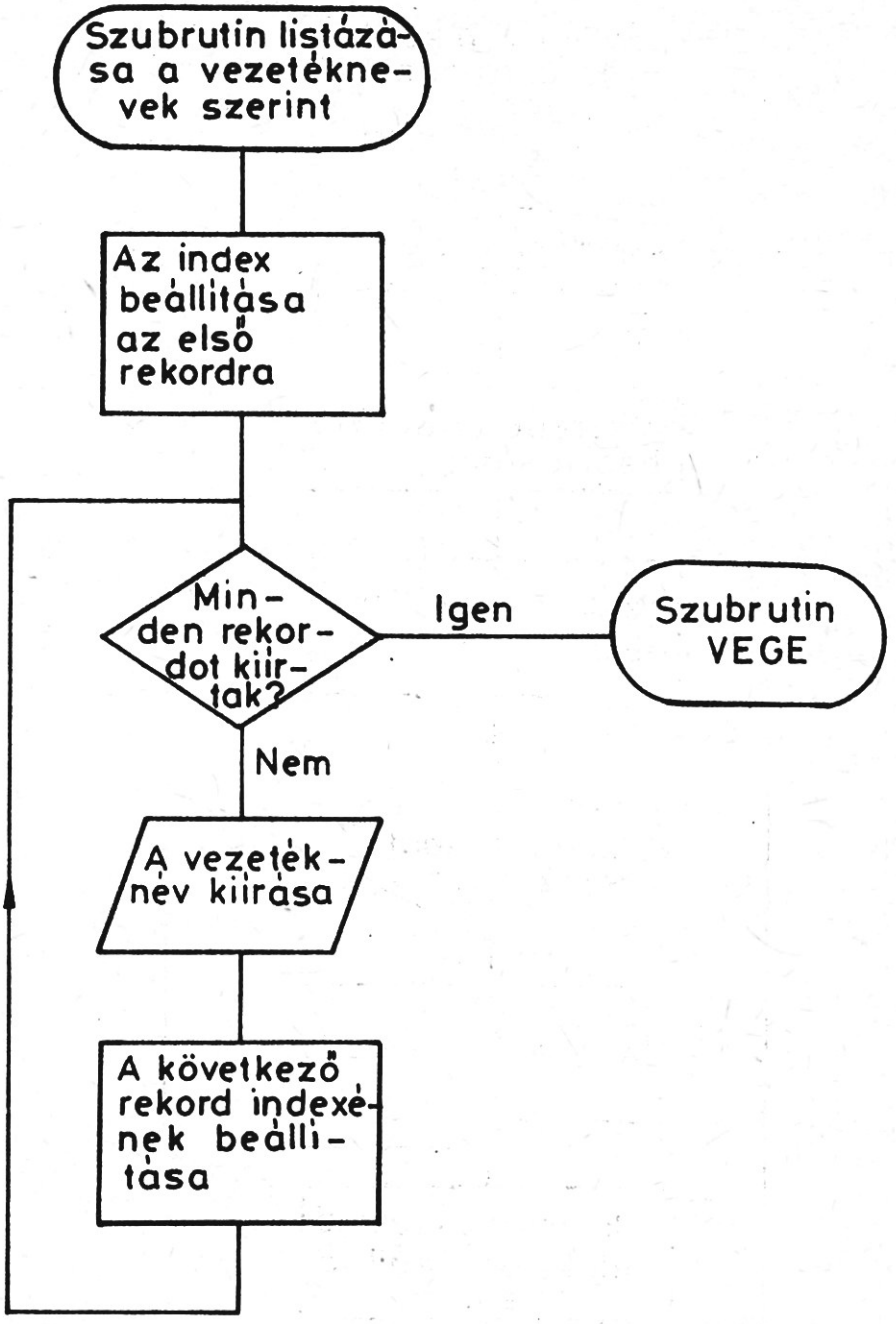


3. SZUBRUTIN: a címek beolvasása

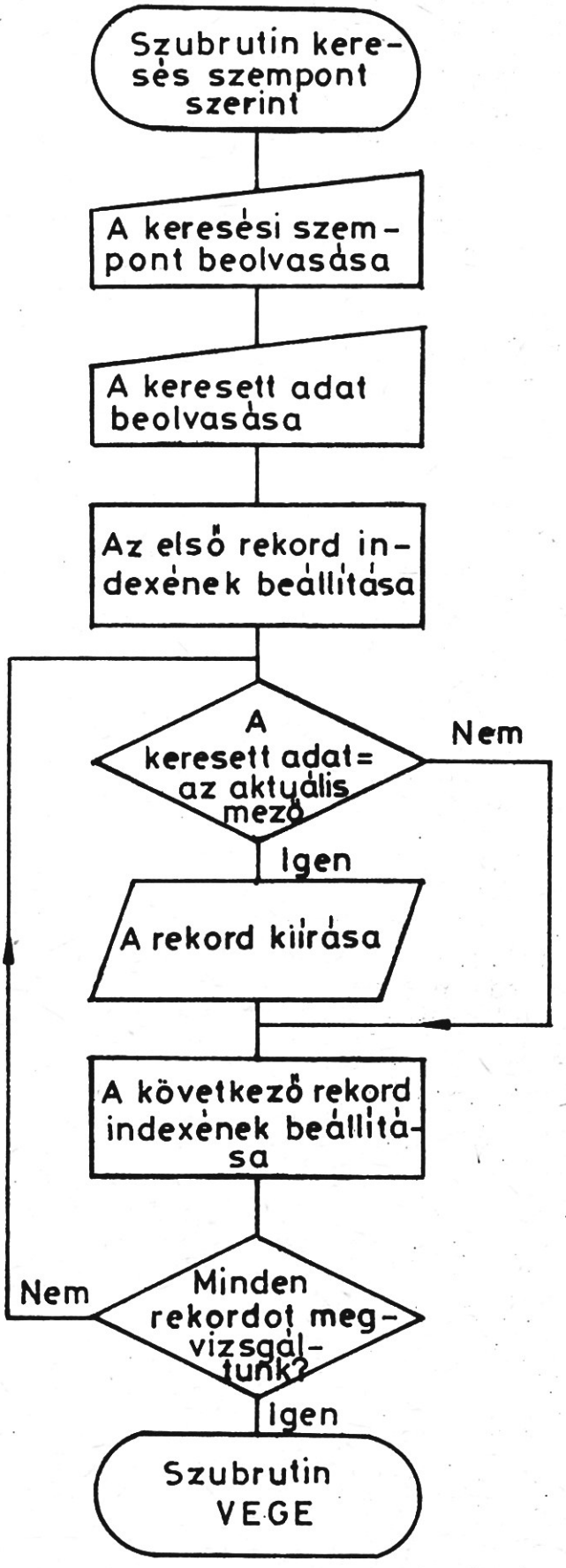



Szubrutin keresése
Lista a vezetéknevek szerint, vagy keresés szempont szerint
A választás beolvasása
Lista a vezetéknevek szerint
Igen
Lista a vezetéknevek szerint
Nem
Keresés szempont szerint
Igen
Keresés szempont szerint
Nem
Érvénytelen választás
Szubrutin VEGE

4/1. SZUBRUTIN: listázás kulcs szerint



4/2. SZUBRUTIN: keresés szempont szerint



Annak ellenére, hogy az utolsó alárendelt folyamatábrát is megrajzoltuk, és ezzel a programtervezést befejeztük, a program begépelését még mindig nem célszerű elkezdeni. Nagyon sok részlet maradt kidolgozatlanul. Még nem terveztük meg pl. az input/output műveletek képernyőmaszkjait, és tulajdonképpen még mindig nem tudjuk, hogyan épül fel a program az elejétől a végéig. Hogyan tervezzük meg a hiányzó részleteket? A következő alfejezetekben ehhez adunk néhány hasznos tanácsot.

1.4 A programírás közvetlen előkészületei

Lehet, hogy az Olvasó már kissé hosszadalmasnak találja a munka előkészítési fázisát, de néhány kísérlet után biztosan egyetért majd velünk abban, hogy alapos tervezés nélkül lehetetlen a hibák elkerülése. Nem a programozás az egyetlen olyan munkaterület, ahol a precíz előkészítés megóvja az embert a többletmunkától, a fölösleges kitérőktől.

1.4.1 A képernyőmaszkok megtervezése

Általános igazság, hogy egy termék kelendőségéhez, használhatóságához nagymértékben hozzájárul külső megjelenési formája, a tetszetős kivitelezés.
Ez alól a programtermékek sem kivételek. Minél világosabb és logikusabb a képernyő felépítése, a használó annál könnyebben kezelheti a programot. A program használója semmit sem tud a programról azon kívül, amit a képernyőn lát. Érthető tehát, hogy egy program „jósága" gyakran a képernyőterveken múlik. A képernyőmaszkok készítését programozástechnikai oldalról a 2.1 alfejezetben tárgyaljuk.
Most azokat a segédeszközöket mutatjuk be, amelyek a tervezéshez szükségesek. A tervezés egyik jól bevált eszköze a tervezőlap. A szabványosított tervezőlap nagyszámítógépes adatfeldolgozásban a programozó nélkülözhetetlen munkaeszköze.
A következő ábrán látható tervlapot kifejezetten Commodore 64-es programok képernyőmaszkjaihoz készítettük. Felosztása megfelel a képernyő 25 × 40-es mezőfelosztásának.
Másoljuk le néhány példányban a tervlapot, és próbáljuk ki a gyakorlatban. Hamarosan látni fogjuk, hogy sokkal gyorsabb és könnyebb így egy tetszetős képernyőmaszkot kialakítani, mintha ugyanazt a programsorok javítgatásával igyekeznénk elérni.

Képernyőmaszk tervezőlap Commodore 64-esre

Jelölések:

Programnév:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
1																																									1
2																																									2
3																																									3
4																																									4
5																																									5
6																																									6
7																																									7
8																																									8
9																																									9
10																																									10
11																																									11
12																																									12
13																																									13
14																																									14
15																																									15
16																																									16
17																																									17
18																																									18
19																																									19
20																																									20
21																																									21
22																																									22
23																																									23
24																																									24
25																																									25
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	

Megjegyzések:

A programkiírás során a kurzor pozícionálása is sokkal egyszerűbb a tervlap sor- és oszlopszámozása alapján. Ráadásul a tervezés közben az is kiderül, hogy egy képernyőhöz a programban hány input/output változót kell használnunk. Így mire végzünk a képernyőtervekkel, a program változólistája (ld. az 1.4.5 alfejezetben) is részben elkészül.

1.4.2 A file-ok megtervezése

Ha a feladat elemzése során eldöntöttük, hogy bizonyos adathalmazokat tárolni fogunk, pontosan meg kell határoznunk az adathalmaz felépítését.
Ezt a munkát is jelentősen megkönnyíti egy célszerűen kialakított tervlap. A következő oldalakon bemutatjuk az általunk készített file-tervlapot, melyen az Olvasó pontosan rögzítheti a programban használt összes file jellemző adatait. Az egyik lapot mintaként kitöltöttük, a másik az Olvasó rendelkezésére áll.

A file-tervlap rovatainak jelentése:

File-név:	Ide kerül az adathalmaz neve. A file nevét ajánlatos úgy megállapítani, hogy azzal utaljunk a benne tárolt adatok jellegére. Pl. egy címeket tartalmazó file neve lehet: CIM, egy pénzügyi adatokat tartalmazó file neve lehet PENZ stb.
Programnév:	A szóbanforgó program neve.
File-típus:	Ebben a rovatban megadjuk a file szerkezetét. A file-típus lehet: soros, relatív, közvetlen elérésű, ISAM, QUISAM stb.
Rekordhossz:	Az adatmezők együttes hossza.
Maximális rekordszám:	Itt jegyezzük meg, ha korlátozott méretű file-al dolgozunk. A file méretét gyakran technikai okok korlátozzák (pl. az adott file-típus megköveteli, vagy a lemez kapacitása korlátozott stb.).
Sorszám:	Az egy rekordhoz tartozó adatmezőket sorszámmal látjuk el.
Típus:	Itt megadjuk az adatmező típusát (numerikus, alfanumerikus, tömörített).
Hossz:	A mező maximális hossza: a benne szereplő karakterek száma.
Pozíció:	A mező kezdete a rekord elejétől számítva.
Változó:	Ebben a rovatban előre rögzíthetjük az input/output változók azonosítóját. Hasznos segítséget jelent majd a változólista felépítésében.

Megjegyzés: Ebben a rovatban felsorolhatjuk a programban használt kódokat (pl. 0=férfi, 1=nő, 2=cég stb.).
A második megjegyzés mezőbe olyan adatokat írhatunk, amelyek az egész file-ra vonatkoznak. (Pl. a program célja az adathalmazzal: ADATRÖGZÍTÉS, KIÉRTÉKELÉS stb.)

Kapcsolat más programokkal: Gyakran előfordul, hogy egy adathalmazt több program használ. Ebbe a rovatba beírhatjuk, hogy mely programok, és milyen célból használják a file-t. FONTOS: ha egy file-t több programban használunk, célszerű minden programban azonos változónevekkel dolgozni. A programcsomag ezáltal áttekinthetőbb, és könnyen módosítható lesz.

FILE-TERVEZŐ LAP A COMMODORE 64-esre

File név: Programnév: Rekordok max. száma:

File típus: Rekordhossz:

REKORDFELÉPÍTÉS-adatmezők

Szám	Megjegyzés	Típus	Hossz	Pozíció	Változó	Jelölés

Megjegyzések:

Kapcsolat más programokkal

Programnév	Hozzáférés módja

FILE-TERVEZŐ LAP A COMMODORE 64-esre

File név: CÍM Programnév: BÉRLET Rekordok max. száma: 300

File típus: REL Rekordhossz: 76

REKORDFELÉPÍTÉS-adatmezők

Szám	Megjegyzés	Típus	Hossz	Pozíció	Változó	Jelölés
1	Rekordszám	N	3	1	A_1	Kulcsok
2	Keresztnév	A	12	4	A_2 $	
3	Név	A	15	16	A_3 $	
4	Status	N	1	31	A_4	1,= teljes, 2,= gyerek
5	Irányítószám	N	4	32	A_5	3,= tanuló, 4.,= nyugdíjas
6	Helység	A	20	36	A_6 $	
7	Utca	A	20	56	A_7 $	
8	Követelés	N	1	76	A_8	0 = Még nincs kifizetve 1 = Ki van fizetve

Megjegyzések:

Bevitel, módosítás, Törlés,

A tagok listájának nyomtatása + címkézés

Kapcsolat más programokkal

Programnév	Hozzáférés módja
	A követelések nyomtatása

1.4.3 Algoritmusok, számítások

Most a tervezésnek egy olyan részletéről próbálunk képet adni, amely fölött a programozók gyakran átsiklanak, és amelynek elhanyagolása nagyon sok hiba forrása lehet. Nagyon fontos ugyanis előre tisztázni, hogy a programban milyen számítási eljárásokat fogunk alkalmazni, milyen képletekkel dolgozunk, pl. egy kerekítő rutinban.
Az alábbiak sok hasznot hozhatnak annak, aki a jótanácsot megszívleli.
Nézzünk példaként egy olyan programot, amely a bruttó bevétel alapján meghatározza a nyereséget:

Módszer	Példa
1. Rögzítsük, hogy mit és miből szándékozunk kiszámítani!	A bruttó bevétel (BB) a vételár (VA) és az eladási ár (EA) közötti 500 Ft-tal csökkentett különbség 10%-a, úgy hogy a vételárat is előzetesen a többletértékadóval csökkentettük.
2. Írjuk fel a számítás menetét képlet formájában!	BB = EA : 1,14–500–VA)*0,1
3. Ellenőrizzük a képlet helyességét kézi számítással vagy zsebszámológéppel!	(5700 : 1,14 − 500 − 2500)*0,1 = 200
4. Ha a képlet helyes, írjuk át BASIC nyelvre! A képletet olyan formában kell BASIC nyelvre átírni, ahogyan az ténylegesen a programba kerül. Ügyeljünk a képletben szereplő változókra. Vizsgáljuk meg, hogy a képlet csak egyszer, vagy többször is előfordul-e majd a programban. Az utóbbi alapján tudjuk eldönteni, hogy szubrutint írjunk-e az eljárásra.	BB = (EA : MI − 500 − VA)*0,1 ahol: EA = eladási ár, beleértve az értéktöbbletadót MI = értéktöbbletadó hányados
5. Írjuk be a képletet a programba, és ellenőrizzük, hogy azonos adatok mellett azonos eredményt ad-e a kézi számítással.	

Az eljárás nagyon hosszadalmasnak tűnhet, de a végső fázisban munkát takaríthat meg. A későbbi programozási hibákat gyakran hibás képletek okozzák, egyszerű számtani műveletek, amiket a programozó az „Alapműveletekben második osztályos korom óta otthonosan mozgok" felkiáltással ellenőrzés nélkül kódolt.

1.4.4 A nyomtatás

A programelőkészítéshez minden esetben hozzátartozik a nyomtatásképek megtervezése is. Ugyanúgy, mint a képernyőmaszknál, itt is fontosak az esztétikai és ergonómiai szempontok, mivel a nyomtatás – ami többnyire lista formájában történik – szintén a kapcsolatteremtés eszköze egyfelől a felhasználó és a program, másfelől a felhasználó és a számítógép között. A nyomtatást az egyszerű listázás kivételével rendszerint mindig a nyomtatni kívánt adatok valamilyen előkészítése előzi meg. Ezért a nyomtatást nagyon gondosan meg kell tervezni.

Ha el akarjuk dönteni, hogy mit milyen formában kell kinyomtatni, a következő szempontokat kell tekintetbe venni:

- mit kell kinyomtatni
- milyen file-okat, milyen rekordokat vagy adatmezőket használunk
- milyen formában kell előkészíteni és kiértékelni az adatokat
- milyen nyomtatón kell nyomtatni.
  A listaképet elsősorban a nyomtató típusa határozza meg lehetőségeivel és tulajdonságaival (nyomtatási kép, nyomtatványszélesség, soronkénti karakterek száma, másolatok száma stb.). Feltehetően az Olvasó csak egyféle nyomtatóval rendelkezik. De ha programjait idegen használók számára írja, akkor a nyomtatást olyan rugalmasan kell megszervezni, hogy többféle nyomtatótípuson is használható legyen. Ez különösen igaz a szabványprogramok esetében.
- milyen legyen a nyomtatáskép?
  Nemcsak azt kell meghatározni, hogy az egyes rekordok mely elemeit és milyen sorrendben kell kinyomtatni, hanem azt is, hogy milyen legyen a nyomtatáskép teljes külalakja. Ide tartozik a nyomtatáskép szélessége, másolatok száma, egysoros/többsoros nyomtatás, feliratok, közbenső feliratok, esetleges összegek és sok más egyéb. A legcélszerűbb, ha ugyanúgy járunk el, mint a képernyőmaszk tervezése során, és a nyomtatásképet először papíron ceruzával tervezzük meg. A képernyőmaszkkal ellentétben egy lényeges különbséget kell figyelembe venni. Az írófejet a képernyőkurzorral ellentétben csak egy irányba tudjuk lekérdezni, ehelyett a programban megfelelő számlálókkal kell dolgozni.
- mi történjen az adatokkal nyomtatás után? (például töröljük a file-t listázás után?)

A nyomtatás megszervezéséhez a 3. fejezetben hasznos rutinokat találnak Olvasóink.

1.4.5 A változólista, ötletek a változók megválasztására

Az a tény, hogy a COMMODORE–BASIC-ben nincs szükség a változók előzetes definiálására, a változók nagyon bőkezű használatára bátorít fel. Sok programozó előzetes tervezés nélkül bátran használja a változókat programjában. Ennek nemcsak az a következménye, hogy olyan áttekinthetetlenné válnak a programok, hogy rövid idő múltán már maga a készítő sem ismeri ki magát bennük. Nehezen fellelhető hibaforrások is keletkeznek, amit többnyire a változók akaratlan kettős használata okoz. A programok MERGE utasítással való összeillesztése vagy az overlay technika alkalmazása (a változó átadása a következő programnak, a részletes leírást lásd a 3. fejezetben) az összehangolatlan változóhasználatnál szinte lehetetlen.
Tehát a változók használatát ésszerűen kell megtervezni. A tervezés segédeszköze a változólista. A változólistának minden felhasznált változóra vonatkozóan a következő információkat kell tartalmaznia:

E$ = Vételár
először a 650-es sorban
a továbbiakban a 670., 830., 835., 1160. sorokban használjuk.

A fenti példa első két sorát, vagyis a változótartalom magyarázatát és annak a sornak a feltüntetését, amelyben a változót először használjuk, minden változólistának feltétlenül tartalmaznia kell. A harmadik sor – azoknak a soroknak a megnevezésével, amelyeken a változó még előfordul – ugyan kívánatos, de nagy ráfordítással jár. Ezért jobb az információt a program megírása után egy kis segédprogram, vagy keresztreferencia táblázat segítségével összeállítani.
Hozzá kell tennünk, hogy ez utóbbi információ feltétlenül hozzátartozik a jó dokumentációhoz és a későbbiekben, a program tesztelésének szakaszában fontos segédeszköze a hibakeresésnek. Az alábbi ötletek a változók használatára ugyan részben kizárják egymást, de önmagukban nagyon hasznosak lehetnek:

– A megfelelően megválasztott változónevek

Sajnos a COMMODORE–BASIC nem tesz lehetővé hosszú változóneveket. De némi fantáziával a két karakterből álló változóneveket is megválaszthatjuk úgy, hogy utaljanak arra, ami a név mögött rejtőzik. Jelöljük pl. az eladási árat EA-val, a vételárat VA-val, a mennyiséget M-mel és a nevet NE$-ral.

– A változó definiálása időnyereséggel jár

A COMMODORE 64-es BASIC értelmezője a numerikus változókat közvetlenül a program mögé a RAM vége felé, a karakteres változókat pedig a RAM végétől a program felé helyezi el az első felhasználás sorrendjében. Az értelmező (interpreter) az elhelyezett változókat minden változóhasználatnál átvizsgálja, egészen addig, amíg a kívánt változót meg nem találja. Így gyakran használt, kritikus változók idejében történő definiálása (a program kezdetén: X$=" ", A=0 stb.) időnyereséget jelenthet.

– A többszöri felhasználással tárhelyet takaríthatunk meg.

Minden változó tárhelyet foglal el. Ha a rendelkezésre álló hely kevés, a változókat különböző feladatokra többször is felhasználhatjuk. Például használhatjuk többször ugyanazt a ciklusváltozót!
Igaz, hogy ez bizonyos esetekben árt a program áttekinthetőségének. Ha a szűk tárkapacitás a változók többszöri használatára kényszerít bennünket, feltétlenül szükséges, hogy pontosan tudjuk, mely változót, mikor, mire használunk.

– Azonos célokra azonos változó

A programban az azonos feladatra lehetőleg mindig ugyanazt a változót használjuk. Ezzel egyrészt sokkal egyszerűbb és áttekinthetőbb lesz a program, másrészt ha programkönyvtárat alakítunk ki, az azonos változóneveket tartalmazó programok nem zavarják egymást, amikor egy programmá fűzzük össze őket.

– A tömbdeklarálással (DIM) tárhelyet takaríthatunk meg

Az indexes változókat a COMMODORE–BASIC-ben csak akkor kell deklarálni, ha 10-nél több elemet tartalmaznak. Az interpreter a deklarálatlan változó számára 10 helyet foglal le. Ha 10-nél kevesebb elemre van szükségünk, a megfelelő DIM utasítással [pl. DIM A$ (4)] a fölösleges helyek lefoglalását megakadályozhatjuk.

1.4.6 A program dokumentációja

Mielőtt a dokumentációról szólnánk, először el kell választanunk egymástól két dolgot, amit ugyanis a programozók gyakran összekevernek. A kezelési útmutatónak – amire egy későbbi fejezetben még kitérünk – semmi köze sincs a program dokumentációjához. A kezelési útmutató a program használója számára készül és így a program használatához szükséges tudnivalókat tartalmazza. Ezzel szemben a dokumentációt a programozó elsősorban saját maga és azok számára írja, akiknek esetleg a programot egyszer tovább kell fejleszteniük vagy módosítaniuk. A dokumentáció a következő részekből áll:

- a program célja és folyamatábrája
- képernyőmaszk-tervlapok
- file-tervlapok
- algoritmusok, képletek
- nyomtatásképek
- változólista
- sorszerinti programleírás

Ésszerű programtervezés esetén a dokumentáció nagyobb része már a program megírása előtt elkészül, kivéve a sorszerinti programleírást. Az utóbbiban hiánytalanul fel kell tüntetni, hogy a program az egyes sorokban milyen műveleteket végez. A programleírás persze csak akkor lesz igazán jó, ha a dokumentációnak ezt a fontos részét a programozó a tulajdonképpeni programozással párhuzamosan végzi. A programot senki sem ismeri olyan jól, mint a programozó a programozás pillanatában. Az időközben érthetetlenné vált listákban később eltéved az ember, sokkal jobb, ha kezdettől fogva dokumentáljuk a programot.
A REM sorok jó kiegészítést jelenthetnek, de nem pótolhatják a sorszerinti programleírást. Ha a programokat könyvtári szubrutinokból építjük fel, ajánlatos az alprogramok részletes leírását szövegszerkesztő programmal elkészíteni. Ez esetben a dokumentációkat ugyanúgy összeszerkeszthetjük, mint a programokat (MERGE).

1.4.7 Gondolatok a program szerkezeti felépítéséről

A program szerkezetének megtervezése a programozás (kódolás) előtti utolsó lépés. A program szerkezetének meghatározására van néhány fontos alapelv. Itt nem annyira a program későbbi használója, mint inkább a programozó szempontjait kell figyelembe venni. A program kifogástalan, áttekinthető felépítése nemcsak a programozást könnyíti meg, hanem a program tesztelését, és mindenekelőtt a későbbi módosításokat és bővítéseket.
A programtervezés alapvetően fontos kérdése a program és a program által kezelendő adatállomány méretének előzetes becslése. A feladat méretezésekor nyilván tekintettel kell lennünk a rendelkezésre álló gép-, ill. a háttértároló kapacitására. Értelmetlen dolog lenne például a Commodore 64-esre egy országos lakónyilvántartási információs rendszert tervezni.
A program méretét persze lehetetlenség hajszálpontosan megmondani. A szükséges változók és tömbök száma és típusa alapján egy hozzávetőleges becslés azonban nemcsak lehetséges, de elengedhetetlenül fontos is. Ha utólag derül ki, hogy túlméreteztük azt az adathalmazt, amit egyszerre kell a tárban kezelnünk, lehet, hogy az egész program felépítését meg kell változtatnunk ahhoz, hogy eredeti elképzelésünket megvalósíthassuk. Így esetleg több napos munkánk kárbavész.

Végül gondoljuk át mégegyszer, milyen korábban megírt szubrutinokat tudnánk a készülő programban felhasználni. Ügyeljünk a változók nevének megválasztására, nehogy az összefűzéskor derüljön ki, hogy az egyes részek azonos változót más-más célra használnak.
Határozzuk meg előre a program várható felosztását. Rögzítsük előre, hogy milyen programsorszámokkal dolgozik a főprogram és az egyes szubrutinok.
Válasszuk el egymástól a szubrutinokat olyan REM sorokkal, amelyek a feladatukra utaló megjegyzéseket tartalmaznak.
A bonyolultabb programokat a legjobb programozók sem tudják egy ültő helyükben megírni. Ezért nagyon fontos mindig pontosan tudni, hogy melyik volt a program legutolsó változata. Célszerű tehát a főprogramot REM sorral kezdeni, például így:

```
10 REM SZAMLAFELDOLGOZAS 2.06-OS VALTOZAT 1984. 12. 10.
```

Ezt a programsort minden alkalommal aktualizáljuk, valahányszor változtatunk valamit a programon.
Ha mindent pontosan a fentiek alapján csinálunk, a tervezésnek ebben a fázisában már igen nagy biztonsággal el tudjuk dönteni, hogy a program valóban elfér-e az adatokkal együtt az operatív tárban.
Még egy jó tanács! Legjobban a hivatásos programozók tudják megmondani, hogy mekkora méreg, amikor egy-egy lényegi programrész befejezéséhez közeledve áramszünet, vagy egyéb váratlan esemény miatt a tár tartalma elvész! Az ilyen bosszúságok elkerülésére mentsük lemezre a már megírt programot vagy programrészt minél gyakrabban, de különösen akkor, amikor egy-egy fontosabb logikai egységgel készen vagyunk!

1.5 A strukturált programozás alapelemei egy példán keresztül: a Commodore 64-es, mint zsebszámológép

A „strukturált programozás" kifejezés az utóbbi időben egyre gyakrabban fordul elő a szakirodalomban. Mit is jelent ez tulajdonképpen?
Ha egy ötlet nyomán valaki világosan meg tudja fogalmazni a programozási feladatot, és a teljes feladatot logikai egységekre tudja bontani, már meg is tette az első lépést a strukturált programozás felé. A logikai egységeket általában szubrutinok formájában valósítjuk meg, de ez nem azt jelenti, hogy csak a szubrutinokból álló program lehet strukturált.
Ha egy egyetlen egységből álló feladatot több, egymástól jól elkülöníthető részfeladatból építünk fel, máris sokkal világosabb és áttekinthetőbb munkát végeztünk, mintha az egyes műveleteket ötletszerű sorrendben raknánk egymás mögé.
Sajnos mégis sokkal többen vannak azok, akik ötletszerűen és nem tervszerűen programoznak.
Sokan terjesztik azt a tévhitet, hogy BASIC nyelven lehetetlenség strukturált programot írni. Az tény, hogy pl. a PASCAL nyelv kifejezetten támogatja, de a BASIC nyelv sem teszi lehetetlenné a strukturáltságot.
Ezt az állításunkat be fogjuk bizonyítani a könyv későbbi fejezeteiben.
Most azonban lássuk a strukturált programozási technikát!

1.5.1 Az előkészítés

A programírás előkészítési szakasza nagyon hasonló ahhoz, amit az 1.4 alfejezetben leírtunk, azzal a különbséggel, hogy minden részletet sokkal pontosabban kell megfogalmaznunk. A teljes feladatot most is elemi részeire szedjük.

Az alapelveket egy példán keresztül mutatjuk be.

1.5.2 A programelemzés

A problémamegoldás első lépése a feladat világos, egyértelmű megfogalmazása. Legyen a mi feladatunk az alábbi:
A Commodore 64-est zsebszámológépként, kalkulátorként szeretnénk használni. Hogy a feladat ne legyen túlságosan egyszerű, ábrázoljuk grafikusan a zsebszámológépet a képernyőn. Egy átlagos zsebszámológép billentyűzete a következő billentyűket tartalmazza*:

I. 0–9 — számbillentyűk
II. +, −, ×, : — az algebrai műveletek billentyűi
III. Az = — a műveletet végrehajtó és az eredményt kifejező billentyű
IV. A C billentyű az utolsó bevitel törléséhez
V. A T billentyű minden regiszter törléséhez (ALL CLEAR)
VI. Az E billentyű számok 10 hatványakénti ábrázolásához
VII. A tizedespont (.) billentyűje
VIII. Az S billentyű a számítógép kikapcsolásához.

* A német rövidítések szerint használjuk a billentyűk elnevezését. (a ford. megjegyzése)

Az egyes billentyűket a Commodore 64-es RETURN billentyűjével működtetjük. Ahhoz, hogy mindig lássuk a képernyőn, hogy éppen mi történik, az aktuális billentyűt inverz (fordított képű) módban ábrázoljuk. A kalkulátor a [−2,93873588E+38, 1.70141183E+38] tartományba eső számokat tudja kezelni. Minden billentyűzött szám látható a kijelzőn, ami 18 karakter hosszúságú üres hely a számológép grafikus ábráján!
Ha hibát vétünk, a program hibajelzést ad és leáll. A műveleti jelek a műveleti billentyűk megnyomásakor megjelennek a bal szélső sarokban, és ezzel egyidejűleg az előző kiírás törlődik. A művelet eredményét az = billentyű megnyomásával írathatjuk a kijelző sávba.
A kalkulátor működési sajátossága, hogy a másodszor beírt számértéket konstansként tárolja és az = billentyű többszöri lenyomására az addigi eredmény és a tárolt konstans között elvégzi többször ugyanazt a műveletet.
A kalkulátor használatának megkönnyítésére a legfontosabb billentyűket (C, T, S, E) a grafikus ábrán megvilágítjuk.

A programot a következő részekre osztjuk fel:

A) Minden változó visszaállítása
B) A kalkulátor grafikus ábrázolása
C) A kiindulási helyzet inverz ábrázolása (S)
D) A billentyűzet lekérdezése GET utasítással
E) A leütött karakterek kiértékelése (S)
F) Kijelzés (S)
G) Vezérlés CRSR billentyűkkel (S)
H) Törlés a C leütésekor (S)
I) Törlés a T leütésekor (S)
J) Műveleti billentyű tárolása és kiírása (S)
K) Az algebrai műveletek elvégzése (S)
L) A kijelző teljes törlése (S)
M) A kijelző törlése (első hely marad) (S)

Az (S)-sel jelölt feladatokból szubrutint készítünk, így a főprogram csak az A, B és D pontokból áll.

1.5.3 A változólista

A programban a következő változóneveket használjuk:

X	=	az akutális képernyőpozíció tárcíme
DY	=	a lenyomott billentyű kódja
I	=	a ciklusváltozó
FLAG	=	ismételt műveletek kapcsolója
RES	=	az algebrai művelet eredménye
Q$	=	a GET lekérdezés változója
Z1$	=	az első beolvasott szám
Z2$	=	a második beolvasott szám
SU$	=	a kijelzés változója
OP$	=	aktuális műveletjel (+, −, × vagy :)

1.5.4 A program felépítése

Most következik a munka dandárja. A tervezett program logikai egységeit még tovább kell bontani elemi programlépésekre. Ha jó munkát végeztünk az első fázisban (a program elemzése során), akkor most nem kell mást tenni, mint az ott meghatározott sorrendben az egyes részeket tovább boncolgatni.
Próbáljuk meg az algoritmust GOTO utasítás nélkül elkészíteni. Az igazi strukturált programot ugyanis arról lehet felismerni, hogy nem tartalmaz GOTO utasítást. Az elágazásokat hacsak lehet, ne GOTO utasítással, hanem az IF...THEN... feltételes elágazással oldjuk meg.

A fentiek alapján a programot a következőképpen építhetjük fel:

1. A tár törlése CLR utasítással
2. A képernyőgrafika elkészítése grafikus karakterekkel
3. Meghívjuk azt a szubrutint, amely a lenyomott billentyűt inverz módban ábrázolja. Hívás előtt az X változót kiindulási értékére kell állítani (X=1868)
4. Lekérdezzük a billentyűzetet GETQ$ utasítással. A program addig várakozik, amíg a gépkezelő érvényes karaktert nem ütött le. Ha a lenyomott billentyű érvényes, meghívjuk a bevitelt kiértékelő szubrutint.
5. Meghívjuk a kiértékelő szubrutint GOSUB utasítással.
6. Itt nem lehet elkerülni a GOTO utasítást (GOTO 4), hiszen a programot ezen a ponton mindig ugyanoda, a billentyűzet lekérdezéséhez kell küldeni.
7. Az összes többi szubrutin munkáját a „Leütött billentyű vizsgálata" programrész szervezi meg.

A szubrutinok felépítése:

A program teljes felépítését áttekintve, rátérhetünk az egyes szubrutinok felépítésére. Azt hogy egy szubrutin maga is szubrutinokat használ, a szövegben (S)-sel jelezzük.

1. SZUBRUTIN: a bevitel kiértékelése

A DY változóban tároljuk az aktuális karakter kódját. Mivel ezt az értéket PEEK(X)-szel közvetlenül a képernyőtárból olvassuk be, figyelembe kell vennünk, hogy az alfanumerikus karakterek kódjai nem azonosak az ASCII kódokkal. Javasoljuk, hogy az Olvasó, mielőtt tovább haladna, nézze át a kézikönyvben az érvényes képernyőkódokat (E. melléklet, 132ff oldal – az angol eredetiben.)
Ebben a szubrutinban azt vizsgáljuk meg, hogy a gépkezelő melyik funkcióbillentyűt nyomta le. A vizsgálathoz öt IF utasítást használunk.

A leütött billentyű:
a) C – (S)
b) T – (S)
c) algebrai műveleti jel – (S)
d) egyenlőség jel – (S)
e) S – a program vége
Ha a leütött billentyű nem azonos a fentiek egyikével sem, akkor az csak számjegy lehet, tehát ki kell írni (S).

A vizsgálat és a megfelelő szubrutin meghívása után ismét vissza kell térni a kiindulási ponthoz, és meg kell vizsgálni a következő billentyűt.
Ezt a szubrutint a program 1000-től 1500-ig terjedő soraiba helyezzük el, és mindig a RETURN billentyű leütése után hívjuk meg.

2. SZUBRUTIN: a gépkezelő a CRSR UP billentyűt nyomta le

Ha az X változó tartalma kisebb, mint 1429, akkor visszatérünk a hívási ponthoz (RETURN). Ezzel elkerüljük, hogy a kurzor a tervezett képből kilépjen.
Ha X tartalma nagyobb, vagy egyenlő mint 1429, akkor az értékét 160-nal csökkentjük, így a kurzor 4 sorral feljebb ugrik. Most azt a karaktert, ahol a kurzor előzőleg állt, POKE utasítással inverzről normál állapotba hozzuk, az újonnan választott karaktert pedig inverz módban ábrázoljuk (S). Ezzel lekezeltük a CRSR UP billentyűt, tehát visszatérhetünk a hívási ponthoz.
A szubrutint a 2000–2500 sorokba helyezzük és mindig a „Melyik billentyű..." szubrutinból hívjuk meg.

3. SZUBRUTIN: A gépkezelő a CRSR DOWN billentyűt nyomta le

A 3-as rutin lényegében ugyanazt a feladatot látja el, mint az előző. A két programrész csak két dologban tér el egymástól:

I. $X \geqq 1429$ helyett ebből a rutinból akkor kell műveletvégzés nélkül a hívási ponthoz visszatérni, ha $X \leqq 1864$.
II. Az X változó tartalmát nem csökkenteni, hanem növelni kell 160-nal, a felfelé ugrás miatt.

A szubrutint a 3000–3500 sorokba helyezzük.

4. SZUBRUTIN: A gépkezelő a CRSR RIGHT billentyűt nyomta le

Az előző két esethez hasonlóan most is ellenőrizzük, hogy a kurzor jobbra léptetéssel a megengedett képben marad-e. A képernyőtár felosztása miatt az ellenőrzés kicsit bonyolultabb, mint az előbb volt. Egy ciklust szervezünk, amelyben az I ciklusváltozó 40-es lépésközökkel (a képernyősornak megfelelően) felveszi a kép jobbszélső pozícióit. A ciklusmagban IF utasítással lekérdezzük, hogy az aktuális (X) pozíció azonos-e valamelyik jobb szélső pozícióval.
Ha igen, akkor műveletvégzés nélkül visszatérünk a hívási ponthoz.
A grafikus ábrán a funkcióbillentyűket egy helyköz választja el a számbillentyűktől, ezért szükség van egy másik ellenőrző ciklusra is, amely megvizsgálja, hogy a kurzort 4 vagy 5 hellyel kell-e jobbra mozgatni.
Ha a ciklus lefut anélkül, hogy közben a feltétel teljesült volna, akkor az X változó 4-gyel nő.
Végül a régi pozíciót ismét töröljük, az újat pedig kivilágítjuk POKE utasítással.
A munka végeztével visszatérhetünk a hívási pontra (RETURN).
A szubrutint a 4000–4500 programsorokba helyezzük és ugyancsak a „Melyik billentyű..." szubrutinból hívjuk.

5. SZUBRUTIN: A gépkezelő a CRSR LEFT billentyűt nyomta le

Az algoritmus ugyanaz, mint az előző szubrutinban volt, csak a ciklusok kezdő és végértékeit kell megfelelően módosítani. A szubrutint az 5000–5500 sorokba helyezzük.

6. SZUBRUTIN: A gépkezelő a C billentyűt nyomta le

A C billentyű hatására törölnünk kell az utolsó bevitelt. Az SU$ és a FLAG változók értékét visszaállítjuk (0 ill. üres füzér), a kijelzést pedig töröljük (S), végül RETURN utasítással kilépünk a szubrutinból.
A szubrutint a 6000–6050 sorokba helyezzük és a „Bevitel kiértékelése" programrészből hívjuk.

7. SZUBRUTIN: A gépkezelő a T billentyűt nyomta le

Visszaállítjuk az összes változó (SU$, OP$, Z1$, Z2$, RES, FLAG) eredeti értékét. Az SU$ füzér értékének visszaállításához meghívjuk a 6-os szubrutint.
A 7. szubrutint a 6100–6150 sorokba helyezzük.

8. SZUBRUTIN: A gépkezelő műveleti billentyűt nyomott le

A beolvasott műveleti jelet az OP$ változóban tároljuk [CHR$(DY)] és PRINT utasítással kiírjuk a bal szélső pozícióra. Az eddig beütött számjegyeket a Z1$ változóban tároljuk, töröljük a kijelzőt és visszatérünk a hívási ponthoz.
A szubrutint a 6200–6250 sorokba helyezzük és a „Bevitel kiértékelése" szubrutinból hívjuk.

9. SZUBRUTIN: A művelet elvégzése

Ezt a szubrutint akkor kell meghívni, ha a gépkezelő az egyenlőségjel billentyűt nyomta le. A Z2$ változóban tároljuk a beolvasott számot, majd az egyenlőségjelet kiírjuk a kijelzőre. IF utasításokkal megvizsgáljuk, hogy milyen műveletet kell elvégezni és a kijelző törlése (S) után az eredményt kiírjuk (S). A szubrutint a 6300–6500 programsorokba helyezzük.

10. SZUBRUTIN: A kijelző törlése

A szubrutin egy ciklusból áll, amelyben a kijelző minden pozícióit POKE utasítással üres karakterekkel töltjük fel. A kijelző a képernyőtár 1188-as címétől az 1207-es címéig terjed, az üres karakter kódja pedig 32.
A szubrutint a 6. és 7. programrészekből hívjuk, és a program 6900–7000-ig terjedő soraiba helyezzük.

11. SZUBRUTIN: A karakterek inverz ábrázolása

Az összes ábrázolható karaktert megjeleníthetjük inverz módban. A inverz karakterek külön képernyőkódokkal rendelkeznek, mégpedig úgy, hogy a inverz karakter kódja mindig 128-cal nagyobb az eredeti karakter kódjánál. A inverz ábrázolást tehát megoldhatjuk egyetlen POKE utasítással.
Ezt a rutint a 2-es, 3-as, 4-es, 5-ös programrészekből hívjuk meg és a programban a 10000–10100-ig terjedő sorokba helyezzük.

12. SZUBRUTIN: Melyik billentyűt nyomta le a gépkezelő?

Ezt a szubrutint a főprogram hívja meg a billentyűzet lekérdezése után. A szubrutin két esetet különböztet meg. Ha a gépkezelő a RETURN billentyűt nyomta le, akkor mindig az 1-es szubrutint, a kurzorbillentyűk lenyomása esetén pedig a megfelelő rutint hívja meg.
Vizsgálat és szubrutinhívás után visszatérünk a hívási ponthoz.
A rutint a 20000–20500-as programsorokba helyezzük.

13. SZUBRUTIN: Kijelzés

A kiírandó számot az SU$ füzérben tároljuk. Visszaállítjuk a Z változó kezdeti értékét (1207), majd meghatározzuk az SU$ hosszúságát és szükség esetén korrigáljuk (maximum 18 karakter).
Egy ciklusban POKE utasítással rendre kiírjuk az SU$-ban tárolt karaktereket (számjegyeket) a megfelelő képernyőpozícióra. A ciklus lefutása után a Z-t ismét 1207-re állítjuk, majd visszatérünk a hívási ponthoz.
A szubrutint a 20100–20300-as sorokba helyezzük.

14. SZUBRUTIN: A kijelző részleges törlése

Úgy töröljük a kijelző tartalmát, hogy a bal szélső pozíción a műveleti jel (+, −, ×, :) megmaradjon. Az algoritmus megegyezik a 10. szubrutin algoritmusával.
A szubrutint a 20500–20600-as sorokba helyezzük.

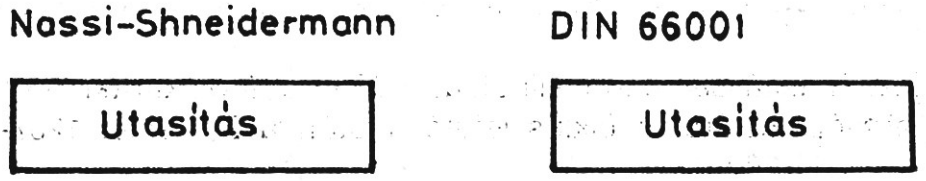
Ezzel a szubrutinok leírásának végére érkeztünk. A tervezett sorszámok megadásával meghatároztuk programunk durva felépítését. Reméljük, hogy az Olvasó saját fejlesztésű programjaiban sikeresen kamatoztatja majd a bemutatott tervezési, programleírási módszert. Lehet, hogy ez az eljárás túlságosan részletesnek tűnik, de mi biztosak vagyunk abban, hogy a befektetett munka a későbbiekben megtérül.
A programelőkészítés írásbeli részének befejezése után próbáljuk meg a programtervet grafikusan ábrázolni. A strukturált programozásnál a folyamatábra helyett célszerű egy másik ábrázolási formát választani: a struktúradiagramot, más néven struktogramot.
A következő oldalakon ismerkedjünk meg ezzel az új grafikus segédeszközzel.

1.6 A struktogram

A struktúradiagramok vagy Nassi–Shneidermann (NS)-diagramok a strukturált programozásban a program ábrázolásánál nagyon hasznos segédeszközök. A folyamatábrákkal szemben a struktúradiagramok előnye, hogy nem használnak adatáramlási vonalakat és nyilakat. Az adatáramlást mindig úgy ábrázoljuk, hogy a végrehajtás sorrendje fentről lefelé irányuljon.
A felhasznált szimbólumok egységes, négyszögű alakjával a teljes adatáramlási tervet egymásba skatulyázhatjuk és így nagyon sok helyet takaríthatunk meg. A következőkben bemutatjuk a hagyományos szimbólumokat. Hogy legyen összehasonlítási alap, mindig felrajzoljuk a DIN 66001-es szabvány szerinti szimbólumot is.

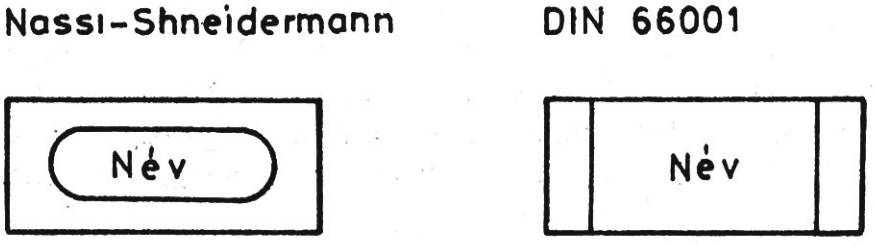
1. A feldolgozási lépés

Egy egyszerű négyszög, amely egy feldolgozási lépésre utal (pontosan ugyanúgy, mint a DIN 66001-es szabványban). A négyszögön belül megadhatjuk szövegesen a végrehajtandó utasítást. Egyetlen szimbólumban több utasítást is elhelyezhetünk, de ez csak akkor célszerű, ha az utasítások azonos vagy hasonló jellegűek (pl. változójelölések). Ha a négyszögben az egyes utasításokat egy vízszintes vonással egymástól elválasztjuk, akkor egy új, finomabb strukturálást végzünk.



2. A szubrutinhívás

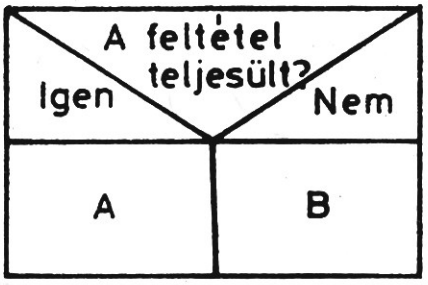
Ezzel a szimbólummal azt jelezzük, hogy valamilyen feladat elvégzésére meghívunk egy szubrutint. A négyszögbe rendszerint csak a megfelelő szubrutin nevét írjuk be. A szubrutin struktúráját itt nem kell részletesebben leírni, ugyanis minden egyes szubrutinhoz saját struktúradiagramot kell készíteni.

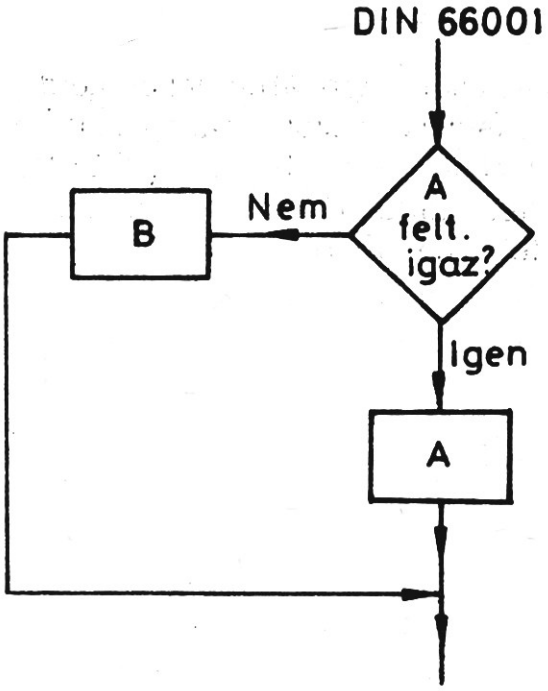


3. A kettős elágazás

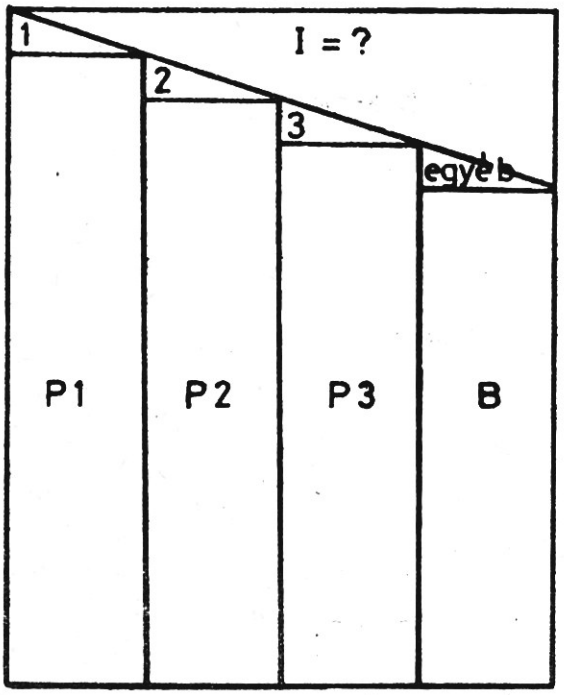
A kettős elágazás annak szemléltetésére szolgál, hogy egy feltétel igaz, vagy hamis voltától függően két különböző utasítást (vagy utasítássorozatot) kell a programban végrehajtani (ez gyakorlatilag megfelel az IF...THEN... utasításnak). Nagyon fontos, hogy valahol a két ág ismét összetalálkozzon.

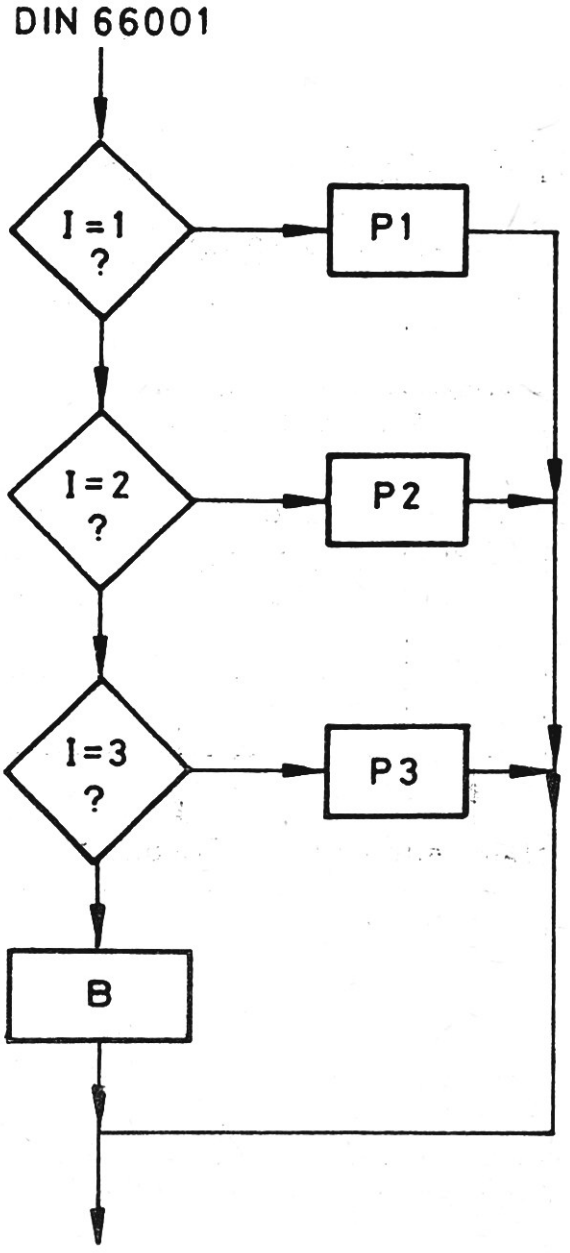
Nassi-Shneidermann





Nassi Shneidermann





4. A többszörös elágazás

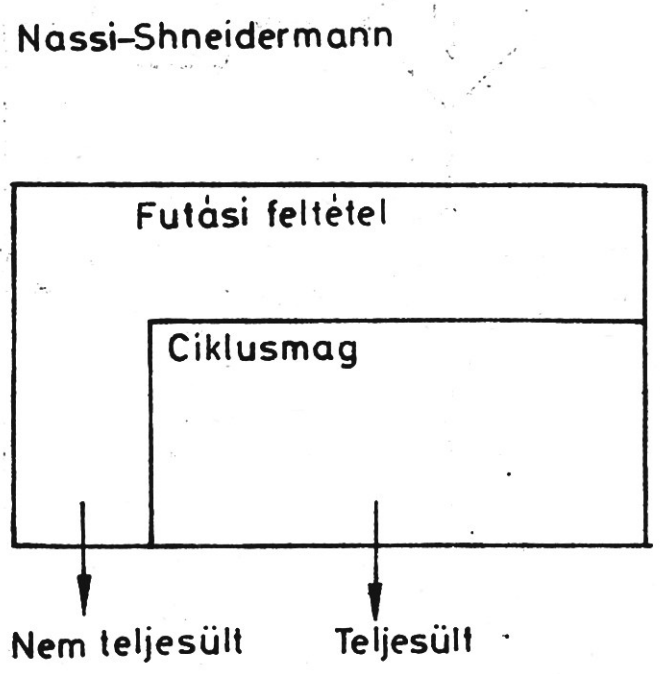
Ha az elágazás csomópontjában feltett kérdés nem eldöntendő kérdés, tehát nem lehet rá igennel vagy nemmel válaszolni, hanem kettőnél több kimenetel lehetséges, akkor többszörös elágazást kell szerveznünk. Ez az eset áll fenn például, ha a végrehajtandó utasítás egy numerikus változó tartalmától függ. Általában a jobb szélső ágat az érvénytelen vagy váratlan változóértékekhez rendeljük hozzá. Az elágazások a szimbólum alsó szélén itt is újra összefutnak. Talán itt ismerhető fel legjobban a struktúradiagramok tulajdonképpeni értelme: a fejlesztő rákényszerül arra, hogy a programelágazásokat a feldolgozás után egy csomóponthoz vezesse vissza. (Ld. a 46. oldal két alsó ábrája.)

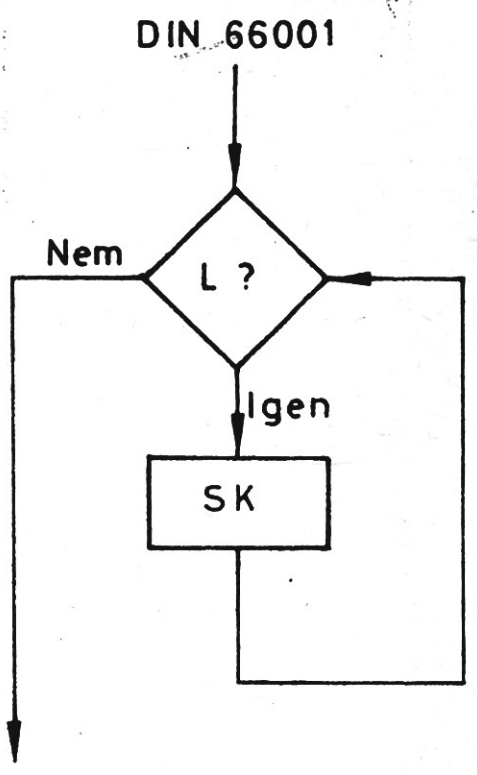
5. A ciklus

A ciklus szervezésére sokféle módszer kínálkozik, de végülis minden ciklus megegyezik két dologban: minden ciklus tartalmaz egy feltételt, és egy lekérdezést (IF), amely megvizsgálja hogy teljesül-e a futási feltétel. Alapvetően kétféle ciklust különböztetünk meg:

A) Előzetes vizsgálattal szervezett ciklus

Ahhoz, hogy a ciklusban elhelyezett utasításokat végrehajthassuk, teljesülnie kell a feltételnek. Az ilyen szervezés megengedi, hogy a ciklust teljesen kikerüljük vagy másképpen: „nullaszor" futtassuk le. Ha a feltétel egyáltalán nem teljesül, a ciklusmag utasításait a program egyszer sem hajtja végre.

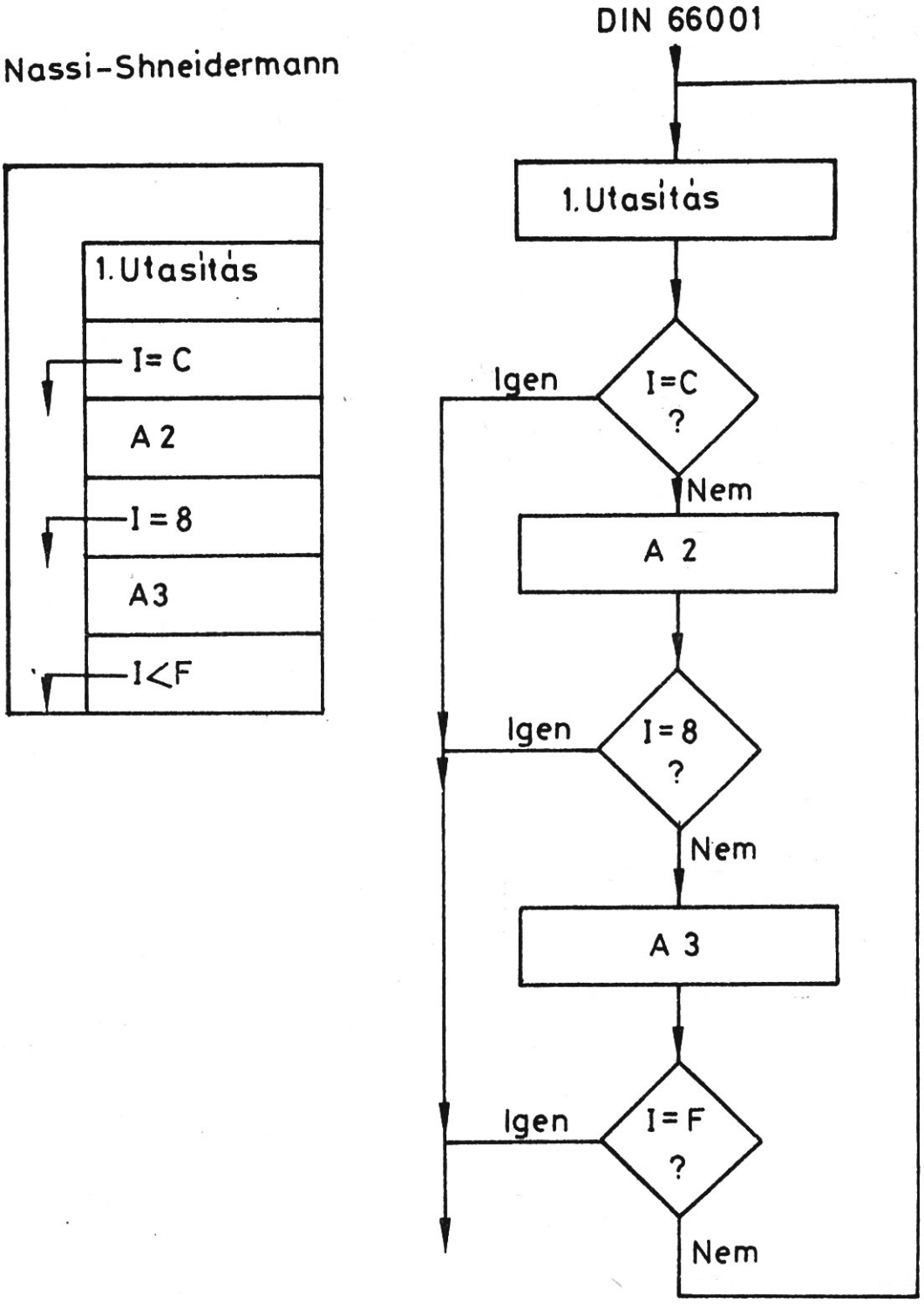




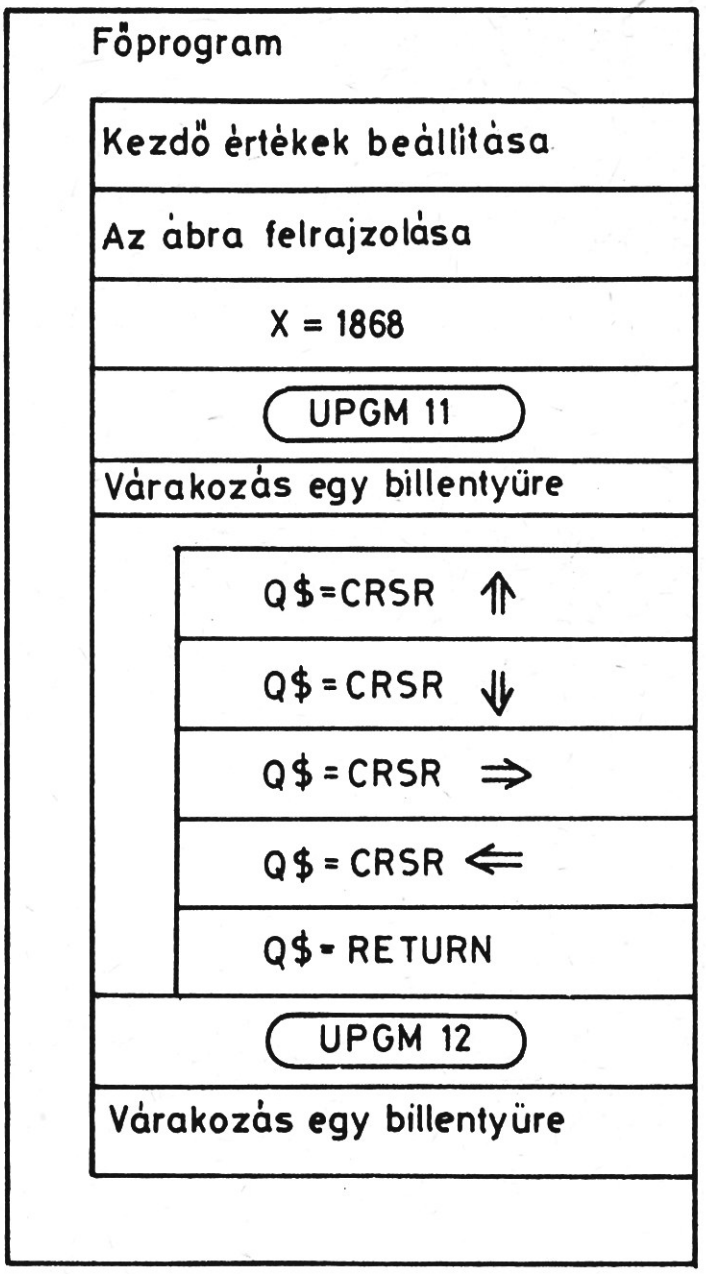
B) Megszakítási feltétellel szervezett ciklus

Ebben az esetben a ciklusmagba megszakítási feltételt vagy feltételeket építünk be. Ha a ciklus lefutása közben valamelyik megszakítási feltétel teljesül, akkor kiugrunk a ciklusból, egyébként újra a ciklusmag elejétől folytatjuk az utasítások végrehajtását.

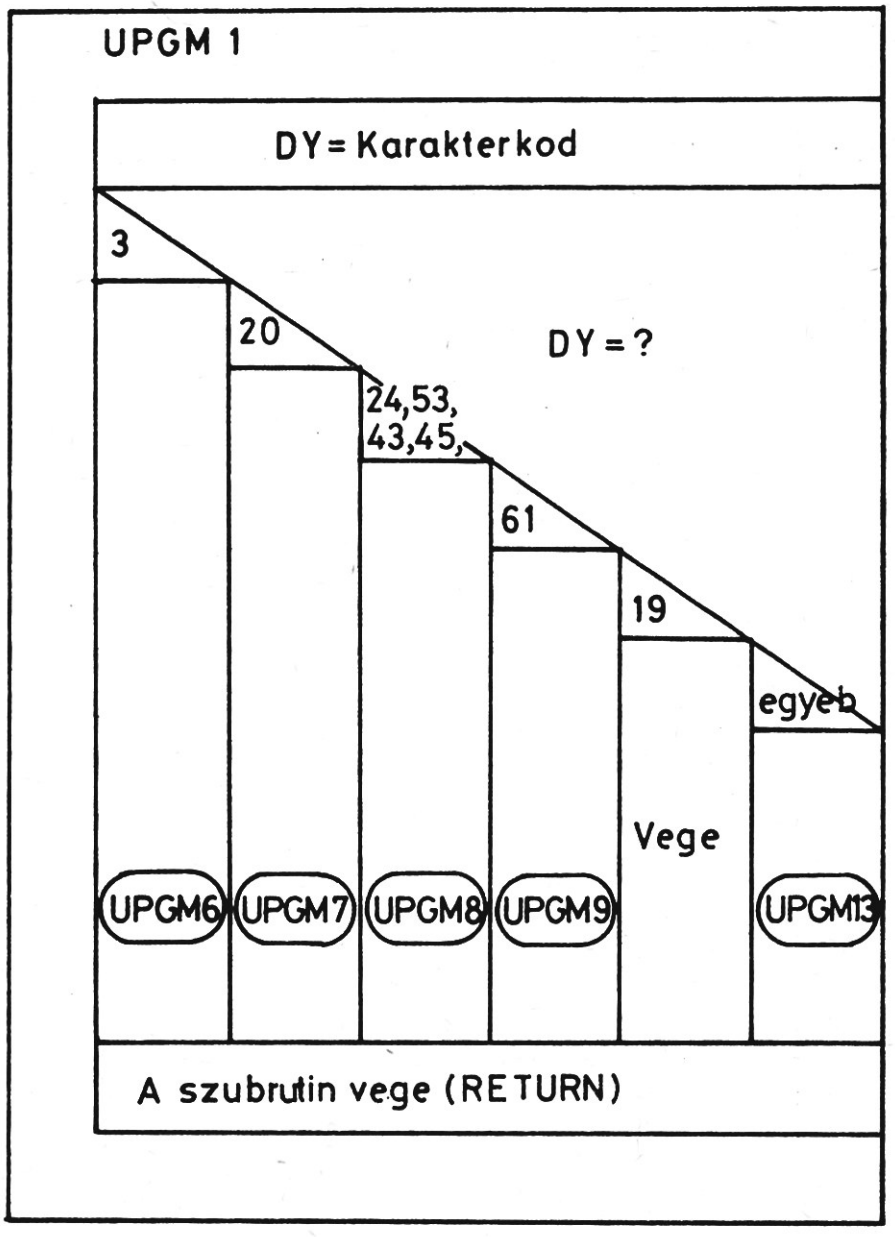
Ha a megszakítási feltételeket rosszul fogalmaztuk meg és esetleg minden feltétel olyan, hogy sohasem teljesül, akkor ún. végtelen ciklus keletkezik és ez általában egy „végtelen" ideig tartó hibakeresés árnyékát veti a programozóra.



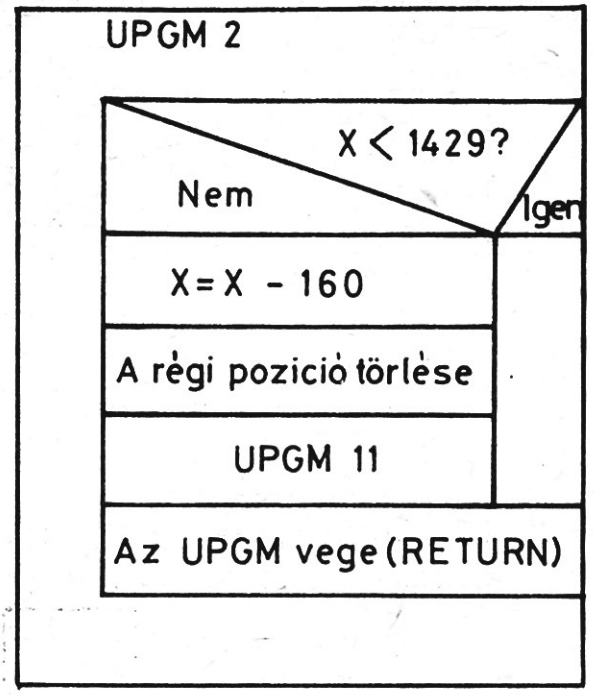
Ennyi felkészülés elegendő is ahhoz, hogy bárki el tudja készíteni önállóan egy program struktogramját.
Egy-egy egység megrajzolása után keretezzük be az egészet, és a bal felső sarokba írjuk be a kidolgozott program vagy szubrutin nevét.
Feltételezzük, hogy az olvasó tettre készen áll saját önálló struktogramjának felrajzolásához. Javasoljuk, hogy mielőtt valóban hozzáfogna, tekintse át a következő oldalakat, ahol teljes részletességgel bemutatjuk a kalkulátor-program struktogramját.


Főprogram
Kezdő értékek beállítása
Az ábra felrajzolása
X = 1868
UPGM 11
Várakozás egy billentyüre
Q$=CRSR ↑
Q$=CRSR ↓
Q$=CRSR →
Q$=CRSR ←
Q$=RETURN
UPGM 12
Várakozás egy billentyüre

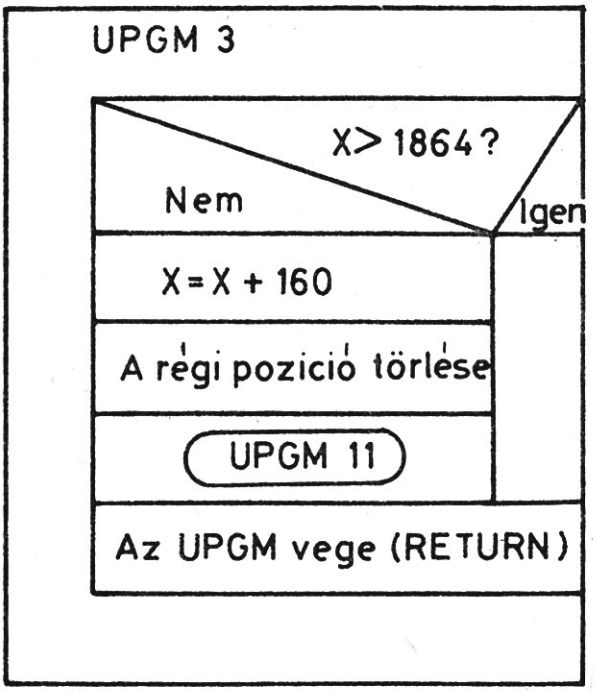
II. 1. SZUBRUTIN: a bevitel kiértékelése

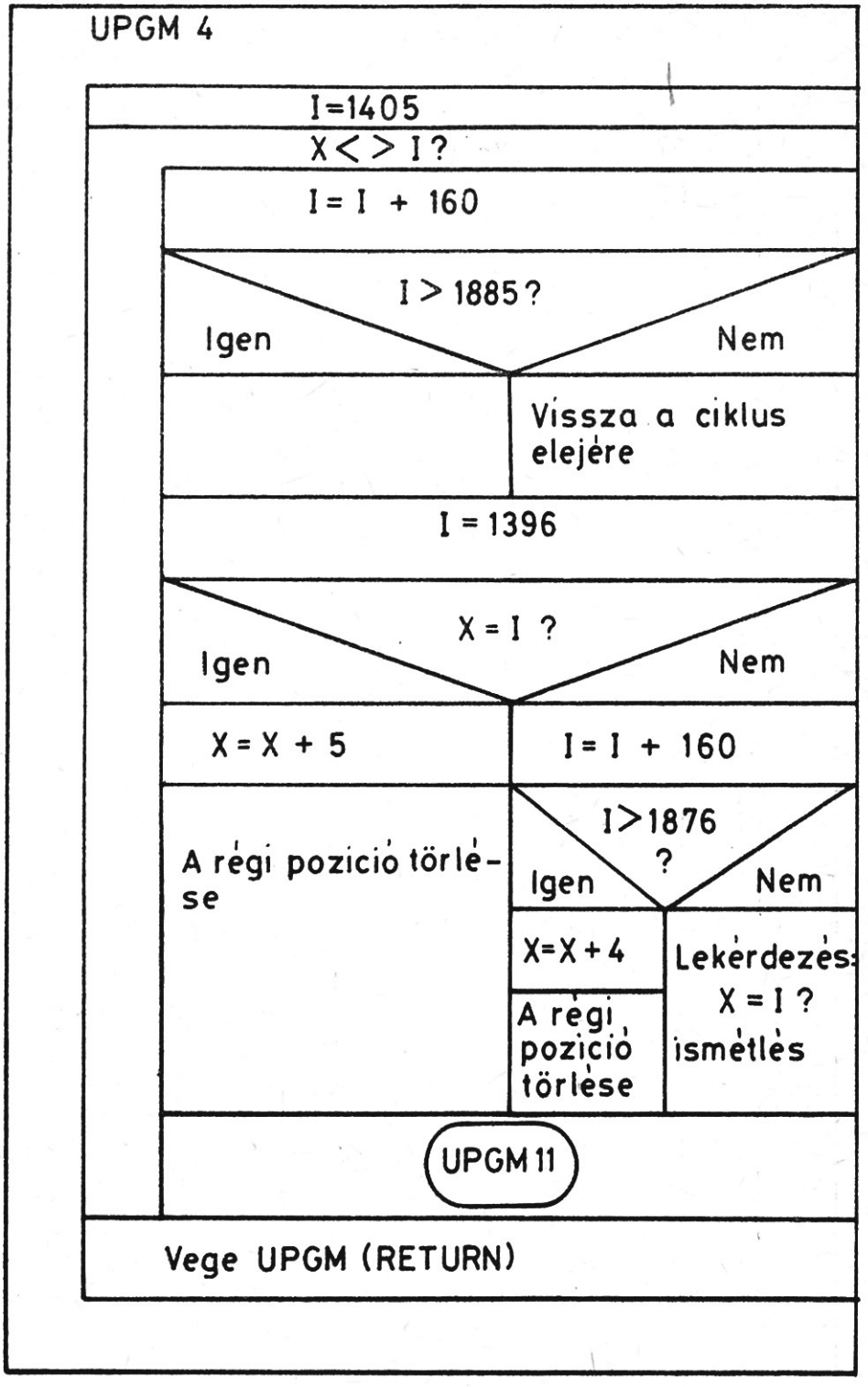


III. 2. SZUBRUTIN: a CRSR ⇧ billentyű lekezelése

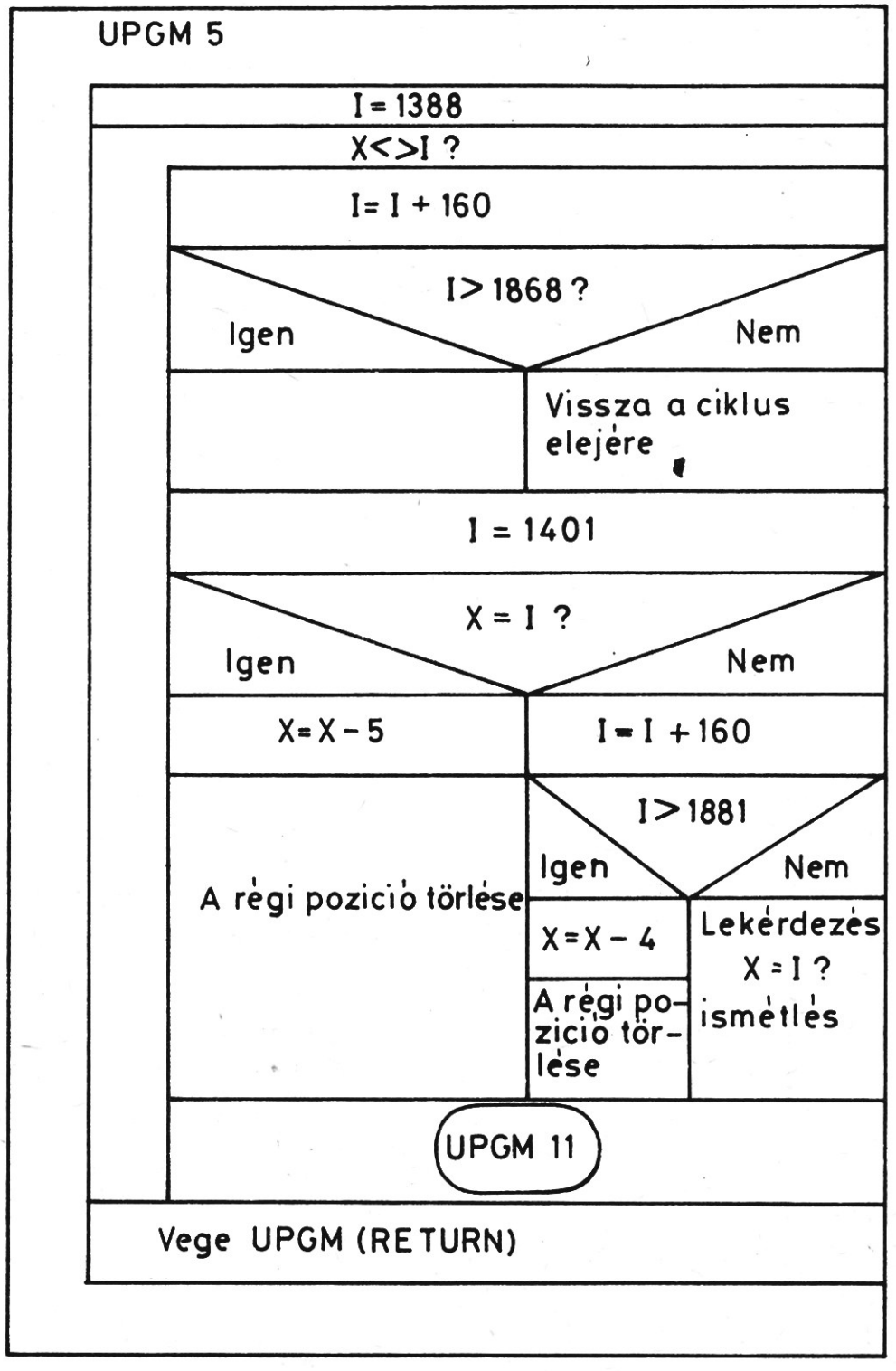


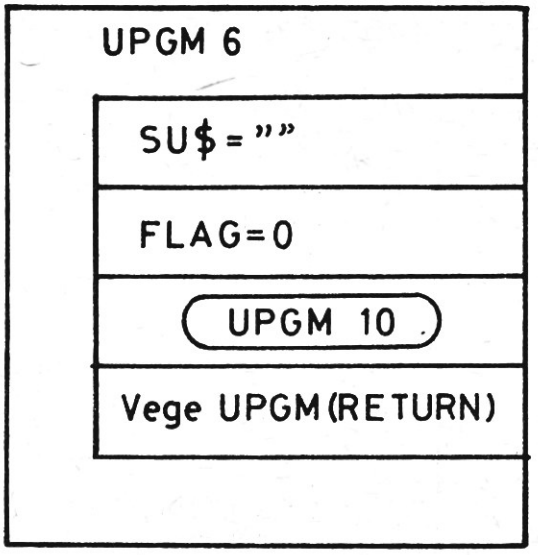
IV. 3. SZUBRUTIN: a CRSR ⇩ billentyű lekezelése

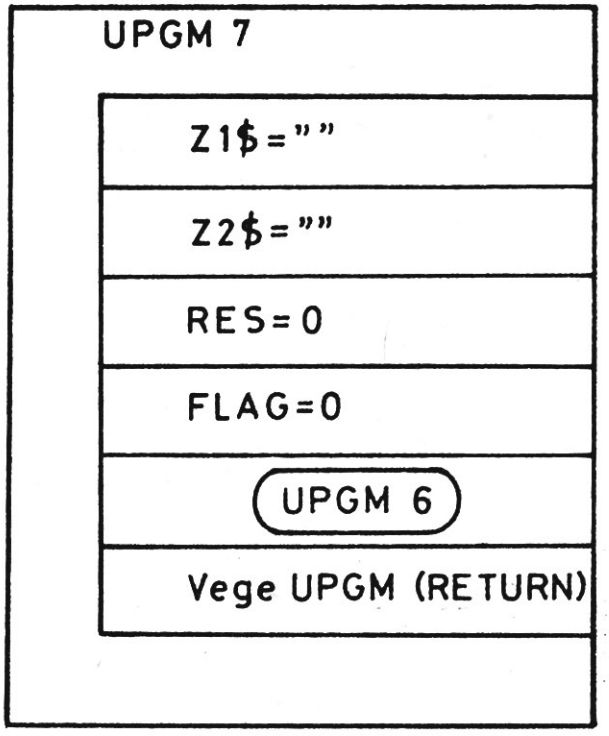


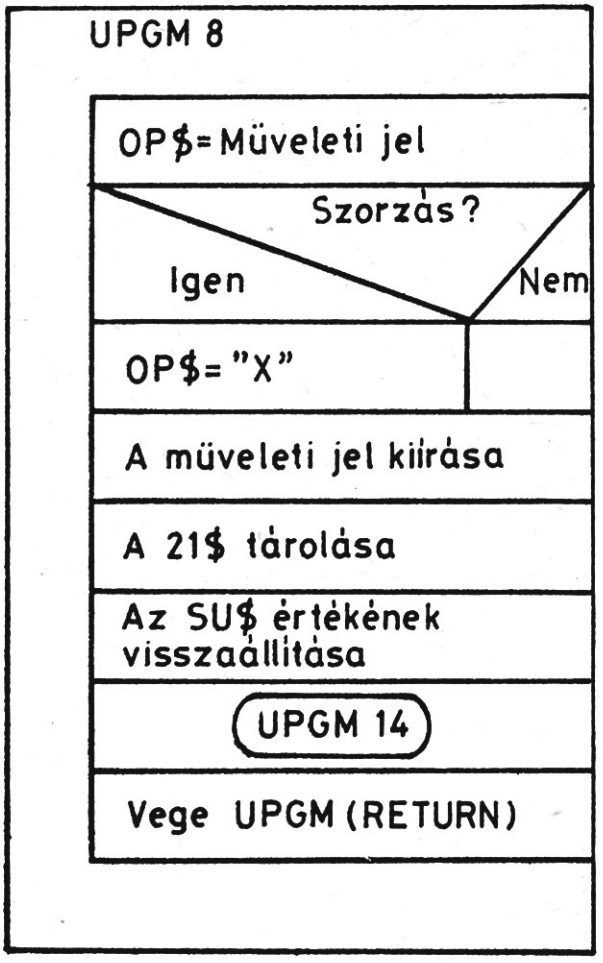

UPGM 4
I=1405
X<>I?
I = I + 160
I > 1885?
Igen
Nem
Vissza a ciklus elejére
I = 1396
X = I ?
Igen
Nem
X = X + 5
I = I + 160
A régi pozició törlése
I>1876 ?
Igen
Nem
X=X+4
A régi pozició törlése
Lekérdezés: X = I ? ismétlés
UPGM 11
Vege UPGM (RETURN)

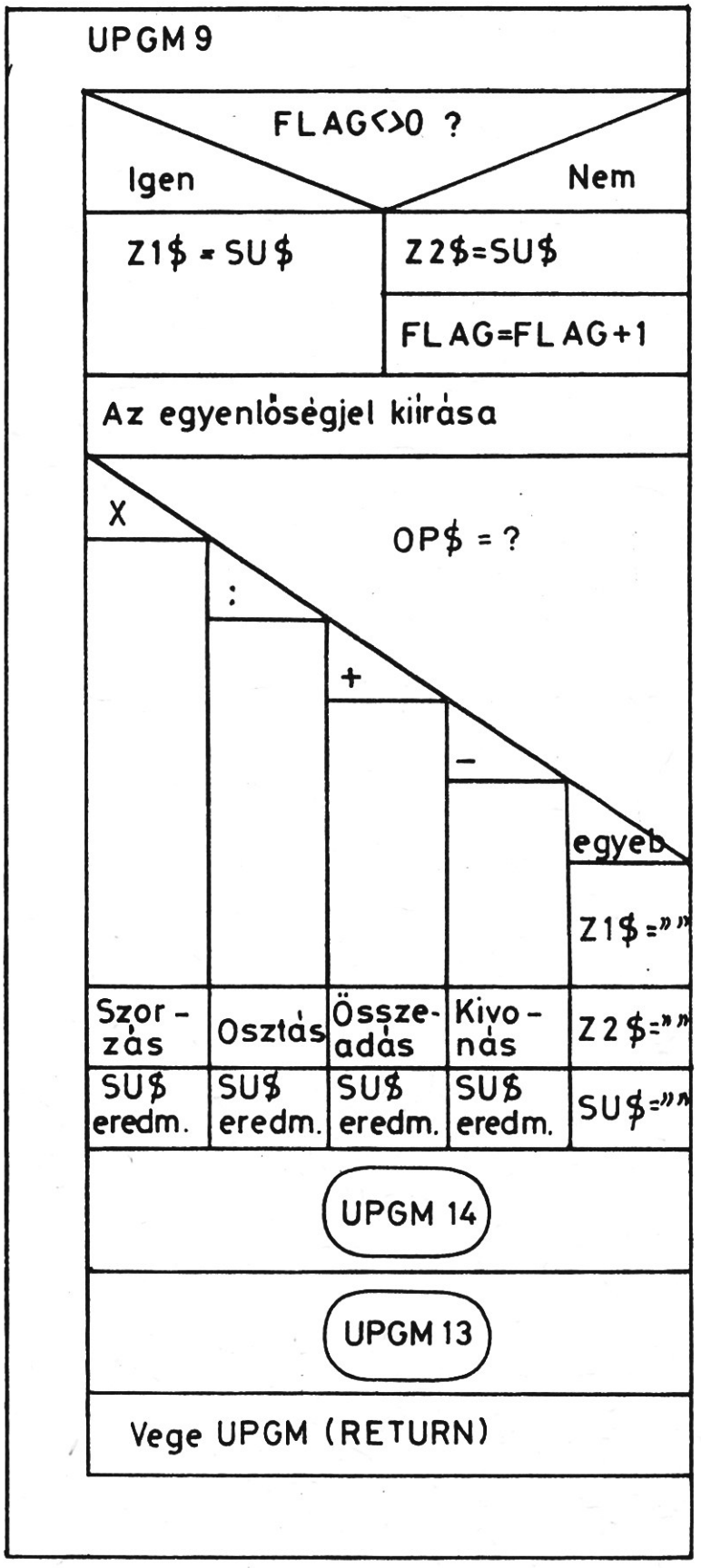
VI. 5. SZUBRUTIN: a CRSR ⇦ billentyű lekezelése



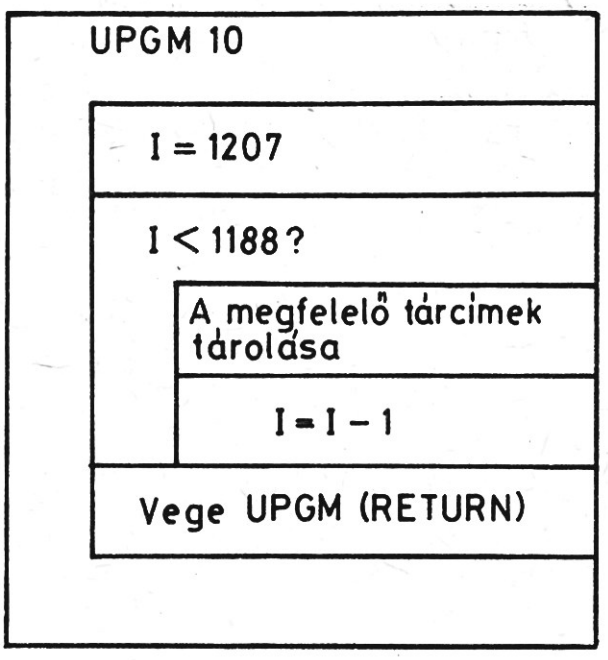

UPGM 6
SU$ = ""
FLAG=0
UPGM 10
Vege UPGM(RETURN)

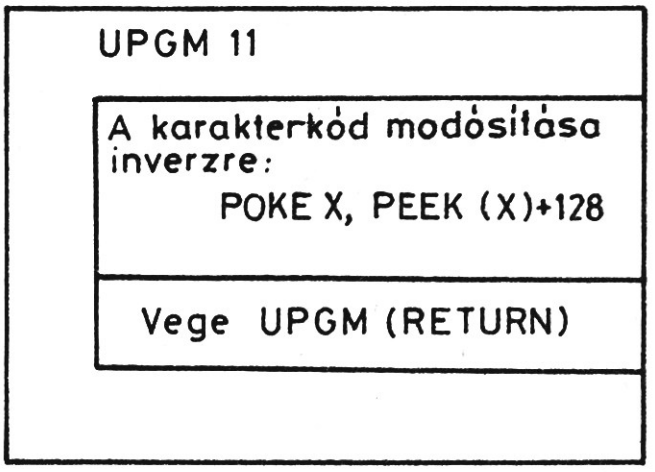

UPGM 7
Z1$=""
Z2$=""
RES=0
FLAG=0
UPGM 6
Vege UPGM (RETURN)

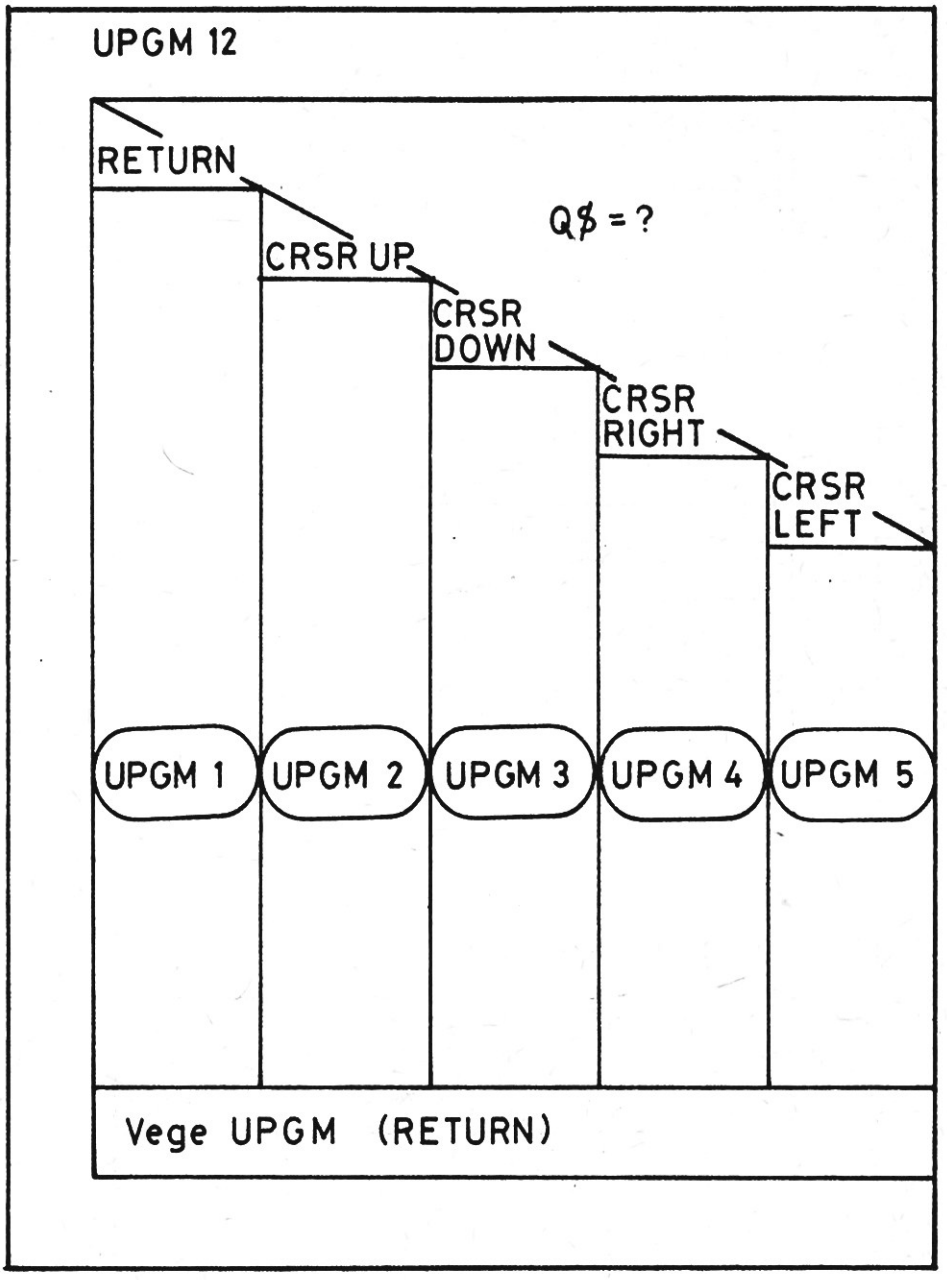

UPGM 8
OP$=Müveleti jel
Szorzás?
Igen
Nem
OP$="X"
A müveleti jel kiirása
A 21$ tárolása
Az SU$ értékének visszaállitása
UPGM 14
Vege UPGM (RETURN)

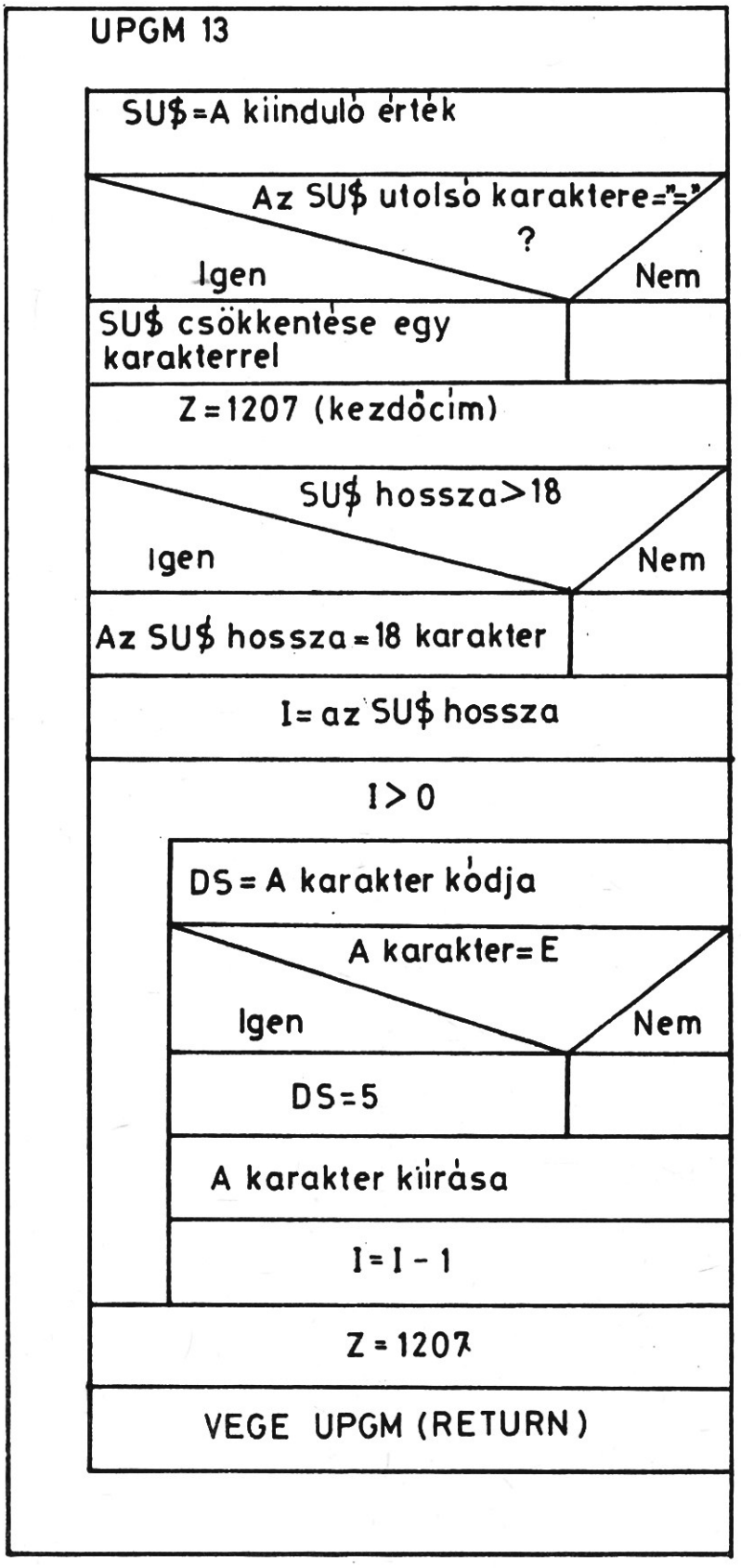

UPGM 9
FLAG<>0 ?
Igen
Nem
Z1$ = SU$
Z2$=SU$
FLAG=FLAG+1
Az egyenlőségjel kiírása
X
:
+
-
OP$ = ?
egyéb
Z1$=""
Szor-zás
Osztás
Össze-adás
Kivo-nás
Z2$=""
SU$ eredm.
SU$ eredm.
SU$ eredm.
SU$ eredm.
SU$=""
UPGM 14
UPGM 13
Vege UPGM (RETURN)

XI. 10. SZUBRUTIN: a kijelző törlése

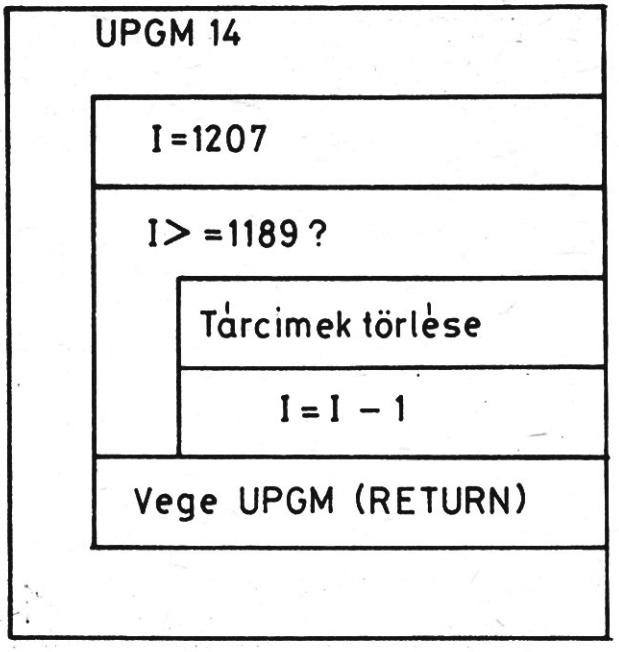



UPGM 11
A karakterkód modósítása inverzre:
POKE X, PEEK (X)+128
Vege UPGM (RETURN)


UPGM 12
RETURN
CRSR UP
CRSR DOWN
CRSR RIGHT
CRSR LEFT
Q$ = ?
UPGM 1
UPGM 2
UPGM 3
UPGM 4
UPGM 5
Vege UPGM (RETURN)


UPGM 13
SU$=A kiinduló érték
Az SU$ utolsó karaktere="="
?
Igen
Nem
SU$ csökkentése egy karakterrel
Z=1207 (kezdőcím)
SU$ hossza>18
Igen
Nem
Az SU$ hossza=18 karakter
I= az SU$ hossza
I>0
DS=A karakter kódja
A karakter=E
Igen
Nem
DS=5
A karakter kiírása
I=I-1
Z=1207
VEGE UPGM (RETURN)

XV. 14. SZUBRUTIN: csak a műveleti jel marad a kijelzőn



1.7 A folyamatábra és a struktúradiagram összehasonlítása

A kétféle módszer tanulmányozása után van alapunk az összehasonlításra. Mint minden új módszer, a struktodiagram is sokakból idegenkedést, kételkedést vált ki.
Az igaz, hogy nincs olyan feladat, amelyet a folyamatábra ne tenne szemléletessé, áttekinthetővé, és a folyamatábra, ahogyan az a nevében is benne van, jól tükrözi a programvégrehajtás folyamatát, az információfeldolgozás menetét, az adatáramlás irányát. Ráadásul kezdetben mindenki, aki eddig folyamatábrával dolgozott úgy érzi, hogy a struktúradiagram elkészítése fáradságosabb a folyamatábráénál. Afelől biztosíthatjuk az Olvasót, hogy ez csak megszokás kérdése. Nézzük meg most alaposan, hogy milyen szempontok alapján tudunk választani a két módszer között, fogalmazzuk meg mindkét módszer előnyeit, és esetleges hátrányait.

A folyamatábra

Kisebb feladatok grafikus ábrázolására kiválóan alkalmas módszer. Ha nagyobb feladattal állunk szemben, ugyan megpróbálhatjuk részfeladatokra bontani, és minden részhez önálló folyamatábrát készíteni, előfordulhat azonban, hogy a munkához nem papírlapra, hanem papírlepedőre lesz szükségünk.

A struktúradiagram

A struktúradiagram, véleményünk szerint egyenesen rákényszeríti az embert arra, hogy strukturáltan gondolkodjék. Készítése kis gyakorlat után egyszerű és gyorsabb a folyamatábra készítésénél. A folyamatábrával szemben ez az ábrázolási mód jóval tömörebb és így kevesebb helyet igényel.
Aki strukturált programokat szeretne írni, annak előbb-utóbb intenzívebben meg kell ismerkednie a struktúradiagramokkal.

1.8 A strukturált BASIC program

Mint korábban már említettük, sok programozó állítja, hogy BASIC nyelven nem lehet strukturált programokat írni. Mi szeretnénk ennek az ellenkezőjét bebizonyítani. Először is lássuk azokat az utasításokat, amelyek a strukturált programozásban kiemelt szerepet töltenek be.

1. A REM

A megjegyzés sorok beillesztésével a különböző programrészeket egymástól vizuálisan jól elválaszthatjuk. Ha a tárkapacitás engedi (és az Olvasó CBM 64-es gépe esetében általában így van!), dokumentáljuk megfelelően a programlistát, ne takarékoskodjunk a REM sorokkal! Minden programrész elejére írjunk legalább egy olyan megjegyzés sort, ami arról ad felvilágosítást, hogy tulajdonképpen mit is csinál az adott programrész.

2. A GOSUB, illetve az ON GOSUB

Használjunk a programban minél több GOSUB utasítást! Ettől egyrészt vizuálisan is áttekinthetőbb lesz a program, másrészt nem jelent gondot a program módosítása sem. Egy szubrutinokból álló programban sokkal könnyebb eligazodni, mint egy tagolatlan programsoráradatban.
Minél kevesebb GOTO utasítást írjunk a programba! A feltétel nélküli ugrásnak semmi keresnivalója sincs a strukturált programban. Minden ugrást, elágazást kössünk valamilyen feltételhez. Így sokkal élesebben láthatóvá válnak a program csomópontjai.
Van ugyan olyan eset, amikor a GOTO utasítás használata nem kerülhető el, de ez lehetőleg a „végső eset" legyen.

A következő oldalakon bemutatjuk a zsebszámológépet imitáló strukturált program listáját. Reméljük, hogy az elmondottakból sikerült meggyőznünk az Olvasót a módszer előnyeiről, és mostantól programot csak strukturált formában hajlandó készíteni.

```
P1

1 REM ************************
2 REM **    A    C B M  64    **
3 REM **  ZSEBSZAMOLOGEPE     **
4 REM **    COPYRIGHT (C)     **
5 REM ** DATA BECKER GMBH     **
6 REM ************************
10 REM VALTOZOK VISSZAALLITASA
20 CLR
95 REM
96 REM
97 REM
98 REM
99 REM
100 REM AZ ABRA FELRAJZOLASA
110 PRINT"^S"
120 PRINT"^Q^Q^Q^Q^Q^]^]*777777777777777777777774
130 FORI=1TO20
140 PRINT"^]^]*                              4
```

```
150 NEXTI
160 PRINT"^]^]*//////////////////////4
170 PRINT"^S^Q^Q^Q^]^]^]U@@@@@@@@@@@@@@@@@@@@I
180 PRINT"^]^]^]J                          J
190 PRINT"^]^]^]J@@@@@@@@@@@@@@@@@@@@K
200 PRINT"^]^]*99999999999999999999994
210 PRINT"^S^Q^Q^Q^Q^Q^Q^Q^Q^]^]^]U@I U@I U@I  U@I U@I
220 PRINT"^]^]^]J7J J8J J9J  JCJ JTJ
230 PRINT"^]^]^]J@K J@K J@K  J@K J@K
240 PRINT"^Q^]^]^]U@I U@I U@I  U@I U@I
250 PRINT"^]^]^]J4J J5J J6J  JXJ J:J
260 PRINT"^]^]^]J@K J@K J@K  J@K J@K
270 PRINT"^Q^]^]^]U@I U@I U@I  U@I U@I
280 PRINT"^]^]^]J1J J2J J3J  J+J J-J
290 PRINT"^]^]^]J@K J@K J@K  J@K J@K
300 PRINT"^Q^]^]^]U@I U@I U@I  U@I U@I
310 PRINT"^]^]^]J0J J.J JEJ  J=J JSJ
320 PRINT"^]^]^]J@K J@K J@K  J@K J@K
330 PRINT"^S^Q^Q";TAB(26);"^RT^R = TELJES TORLES
340 PRINTTAB(26);"^RC^R = TORLES
350 PRINTTAB(26);"^RE^R = KITEVO"
360 PRINTTAB(26);"^RS^R = VEGE"
370 PRINT"^S^Q^Q^Q^Q^Q^Q^]^]^]^]^]^]^R     DATA BECKER CBM 64      ^R"
395 REM
396 REM
397 REM
398 REM * KEPERNYOCIM AZ X-BEN
399 REM * TAROLAS ES A KARAKTEREK
400 REM * AZ INVERZ ABRAZOLASA
410 X=1868
420 GOSUB10000:REM REVERSE ABRAZOLAS
495 REM
496 REM
497 REM
498 REM
499 REM
500 REM * A BILLENTYUZET LEKERDEZESE
510 GETQ$:IFQ$<>"^Q"ANDQ$<>"^Q"ANDQ$<>"^]"ANDQ$<>"^J"ANDQ$<>CHR$(13)THEN510
520 GOSUB20000
530 GOTO510
995 REM
996 REM
997 REM
998 REM
999 REM
1000 REM * A BEVITEL KIERTEKELESE
1010 DY=(PEEK(X)-128):REM KARAKTERKOD DY TAROLASA
1020 IFDY=3THENGOSUB6000:RETURN
1030 IFDY=20THENGOSUB6100:RETURN
1040 IFDY=24ORDY=58ORDY=43ORDY=45THENGOSUB6200:RETURN
1050 IFDY=61THENGOSUB6300:RETURN
1060 IFDY=19THENPRINT"^S":END
1070 GOSUB20100:REM KIIRAS
1080 RETURN
1995 REM
1996 REM
1997 REM
1998 REM
1999 REM
2000 REM * A FEL BILLENTYU LEKEZELESE
2010 IFX<1429THENRETURN
2020 X=X-160:POKEX+160,PEEK(X+160)-128:GOSUB10000:RETURN
2995 REM
2996 REM
2997 REM
```

```
2998 REM
2999 REM
3000 REM * A LE BILLENTYU LEKEZELESE
3010 IFX>1864THENRETURN
3020 X=X+160:POKEX-160,PEEK(X-160)-128:GOSUB10000:RETURN
3995 REM
3996 REM
3997 REM
3998 REM
3999 REM
4000 REM * A JOBBRA BILLENTYU LEKEZELESE
4010 FORI=1405TO1885STEP40
4020 IFX=ITHENRETURN
4030 NEXTI
4040 FORI=1396TO1876STEP40
4050 IFX=ITHENX=X+5:POKEX-5,PEEK(X-5)-128:GOSUB10000:RETURN
4060 NEXTI
4070 X=X+4:POKEX-4,PEEK(X-4)-128:GOSUB10000:RETURN
4995 REM
4996 REM
4997 REM
4998 REM
4999 REM
5000 REM * A BALRA BILLENTYU LEKEZELESE
5010 FORI=1388TO1868STEP40
5020 IFX=ITHENRETURN
5030 NEXTI
5040 FORI=1401TO1881STEP40
5050 IFX=ITHENX=X-5:POKEX+5,PEEK(X+5)-128:GOSUB10000:RETURN
5060 NEXTI
5070 X=X-4:POKEX+4,PEEK(X+4)-128:GOSUB10000:RETURN
5995 REM
5996 REM
5997 REM
5998 REM
5999 REM
6000 REM * A C BILLENTYU LEKEZELESE
6010 SU$="":FLAG=0
6020 GOSUB6900:RETURN
6095 REM
6096 REM
6097 REM
6098 REM
6099 REM
6100 REM * A T BILLENTYU LEKEZELESE
6110 Z1$="":Z2$="":RES=0:FLAG=0:GOSUB6000:RETURN
6195 REM
6196 REM
6197 REM
6198 REM
6199 REM
6200 REM * A MUVELETI JEL BILLENTYU LEKEZELESE
6210 OP$=CHR$(DY):IFASC(OP$)=24THENOP$="X"
6220 PRINT"^S^Q^Q^Q^Q^]^]^]^]";OP$
6230 Z1$=SU$:SU$="":GOSUB20500
6240 RETURN
6295 REM
6296 REM
6297 REM
6298 REM
6299 REM
6300 REM * A MUVELET ELVEGZESE
6310 IFFLAG<>0THENZ1$=SU$:GOTO6330
6320 Z2$=SU$:FLAG=FLAG+1
6330 PRINT"^S^Q^Q^Q^Q^]^]^]^]";"="
```

```
6340 IFOP$="X"THENRE=VAL(Z1$)*VAL(Z2$):GOSUB20500:SU$=STR$(RE):GOSUB20100:RETURN
6350 IFOP$=":"THENRE=VAL(Z1$)/VAL(Z2$):GOSUB20500:SU$=STR$(RE):GOSUB20100:RETURN
6360 IFOP$="+"THENRE=VAL(Z1$)+VAL(Z2$):GOSUB20500:SU$=STR$(RE):GOSUB20100:RETURN
6370 IFOP$="-"THENRE=VAL(Z1$)-VAL(Z2$):GOSUB20500:SU$=STR$(RE):GOSUB20100:RETURN
6380 Z1$="":Z2$="":SU$="":GOSUB20500:GOSUB20100:RETURN
6895 REM
6896 REM
6897 REM
6898 REM
6899 REM
6900 REM * A KEPERNYO TORLESE
6910 FORI=1207TO1188STEP-1
6920 POKEI,32
6930 NEXTI
6940 RETURN
9995 REM
9996 REM
9997 REM
9998 REM
9999 REM
10000 REM * A KARAKTER REVERSE ABRAZOLASA
10010 POKEX,PEEK(X)+128
10020 RETURN
11195 REM
11196 REM
11197 REM
11198 REM
11199 REM
20000 REM * CSAK A MUVELETI JEL LATHATO
20010 IFQ$=CHR$(13)THENGOSUB1000:RETURN
20020 IFQ$="^Q"THENGOSUB2000:RETURN
20030 IFQ$="^Q"THENGOSUB3000:RETURN
20040 IFQ$="^]"THENGOSUB4000:RETURN
20050 IFQ$="^J"THENGOSUB5000:RETURN
20095 REM
20096 REM
20097 REM
20098 REM
20099 REM
20100 REM * KIIRAS
20110 SU$=SU$+CHR$(DY):IFRIGHT$(SU$,1)="="THENSU$=LEFT$(SU$,LEN(SU$)-1)
20120 Z=1207
20130 IFLEN(SU$)>18THENSU$=RIGHT$(SU$,18)
20140 FORI=LEN(SU$)TO1STEP-1
20150 DS=ASC(MID$(SU$,I,1))
20160 IFMID$(SU$,I,1)="E"THENDS=5
20170 POKEZ,DS
20180 Z=Z-1
20190 NEXTI
20200 Z=1207
20210 RETURN
20495 REM
20496 REM
20497 REM
20498 REM
20499 REM
20500 REM
20510 FORI=1207TO1189STEP-1
20520 POKEI,32
20530 NEXTI
20540 RETURN

READY.
```

2. FEJEZET
A PROGRAM RÉSZEI

Minden programnak, álljon mindössze egy REM sorból, vagy egyetlen

```
10 GOTO 10
```

sorból, választ kell adnia három kérdésre:

MIT (kell tenni)? MIVEL (milyen adatokkal)? HOVÁ (kerüljön a végeredmény)?

Ha jól megnézzük, ennek a kritériumnak még az alábbi, rendkívül egyszerű programsor is eleget tesz:

```
10 PRINT "HALLO"
```

MIT KELL TENNI?... nyomtatni. MIVEL?... a HALLO szöveggel. HOVÁ?... a képernyőre.

Általában természetesen a feldolgozás tárgya nem a programban elhelyezett szöveg – mint ebben az egyszerű példában – hanem egy bizonyos adathalmaz, amit valamilyen úton be kell tölteni a tárba (pl. lemezről vagy billentyűzetről).

A három kérdés tehát a következő három számítógépes tevékenységre utal:

MIVEL (milyen adatokkal	adatbevitel
MIT	adatfeldolgozás
HOVÁ	adatkiírás

A három kérdés kiváló támpont a programozási részfeladatok csoportosításához. Gyakorlatilag minden program alapvetően ezt a három tevékenységet végzi el, információt kap, azt feldolgozza és az eredményt kiírja.
A következőkben mindhárom területről bemutatunk néhány ötletet és megoldást, azzal a szándékkal, hogy az Olvasó ezeket mint építőköveket felhasználhassa saját programjaiban, de főként olyan programokban, amelyek nem házi használatra, hanem professzionális célokra készülnek.
A könyvben bemutatott szubrutinok változóit úgy választottuk meg, és a rutinokat is úgy sorszámoztuk, hogy minél könnyebben illeszthetőek legyenek tetszőleges programokhoz. Használhatóságukat azzal is igyekeztünk növelni, hogy legtöbbjük paraméterezhető, így az adott változó tartalmának módosításával tetszőleges adathalmazra alkalmazhatóak.
A jobb megértés kedvéért a 2. fejezetben végig azonos mintafeladaton mutatjuk be a leírtakat, nevezetesen egy címfile rendszeren, amely ráadásul változtatható hosszúságú adatokkal dolgozik.
Ha az Olvasó egy programmá illeszti össze, és megfelelően paraméterezi a szubrutinokat, máris rendelkezésére áll egy kis rendszer, amelyet kedve szerint használhat pl. hanglemezei, könyvei stb. nyilvántartására.

2.1 Az adatbevitel

Hogy mi mindent lehet a fenti cím alatt érteni, azt a DATA BECKER Fejlesztési Osztály vezetőjeként tevékenykedő szerző mindennapi gyakorlatából vett példa mutatja: Bizonyos problémák tisztázása céljából felkeresett bennünket egy vevő. Annak érdekében, hogy gyorsan segíthessünk neki, rögtön magával hozta programjait is. A programok adatok statisztikai kiértékelésére készültek.
Nem kis meglepetésünkre a mintegy 15 program logikája teljesen azonos volt, egymástól csak a képletekben szereplő paraméterekben különböztek, és a paraméterek minden programban konstansként voltak beépítve!
Az Olvasó már biztosan kitalálta, hogy a 15 programot egyetlen programmal lehetett volna helyettesíteni, amely az éppen aktuális paramétereket klaviatúráról beolvassa, a számításokat elvégzi, az eredményeket pedig kiírja nyomtatóra vagy lemezre, attól függően, hogy mire akarjuk a továbbiakban használni.
A következő fejezetben pontosan erről lesz szó. Elsőként a billentyűzetről történő adatbevitelt tárgyaljuk.

2.1.1 A képernyőmaszk

Képernyőmaszk alatt egy rögzített képernyőfelépítést értünk, amely rögzített adatmezőket tartalmaz.
Ezt úgy is tekinthetjük, mint egy sablont vagy űrlapot, amelyen az egyes rubrikákat ki lehet tölteni.
A jó képernyőmaszk a programozó névjegykártyája, hiszen a gépkezelő a programból ezen kívül semmit sem lát.
A program használata során a gépkezelővel minden pillanatban tudatni kell, hogy mikor milyen tevékenységet várnak tőle, pl. billentyűt üssön le stb. Az egyértelműség különösen fontos adatbevitel közben, hiszen a bevitt adatokon múlik az eredmény minősége. A képernyőmaszk a hibátlan adatbevitel megszervezésének eszköze. Ezért nagyon fontos, hogy pl. egy megadott programciklus mindig azonos képernyőmaszkkal dolgozzon.

A képernyőmaszk alapvetően háromféle mezőből áll.

1) A fejlécből – amely minden pillanatban megmutatja, hogy a program a feldolgozásnak melyik stádiumában van. Ha lemezen tárolt adathalmazokkal dolgozunk, akkor ebben a mezőben kiírhatjuk az aktuális file nevét.
2) A beviteli mezőből – amely a feladat szerkezetétől függően több egységből állhat, és amelyből a kurzor csak helyes adatbevitel után léphet ki.
3) A lábjegyzetből – ami közli a gépkezelővel, hogy az adott programrészből hogyan léphet ki, ha egyébként az adatbeviteli mód a kilépést nem teszi lehetővé.

A képmaszk szerkesztésénél lényeges kérdés a színek megválasztása. A Commodore 64-es a maszkfelépítésnek bűbájos lehetőségeit kínálja. Különösen akkor, ha színes televízióval (monitorral) dolgozunk. Ügyeljünk azonban arra, hogy a jóból is megárt a sok. Minden esetben figyelembe kell venni az ésszerűség szempontjait, nehogy ügyetlen színválasztással a szemet irritáljuk, vagy túlságosan kifárasszuk. A képernyőmaszk nem műalkotás, amivel a

programozó imponál, hanem munkaeszköz a későbbi gépkezelők számára, akik adott esetben nap mint nap több órán át dolgoznak a képernyő előtt.
A hatásvadászat és a színdömping többnyire bumerángnak bizonyul, ami előbb-utóbb visszaüt a programozó fejére. Ha nem színes televízióval dolgozunk, hanem egy zöld/fekete monitorral, akkor is érdemes a színekkel kísérletezni, ugyanis a színek különböző megvilágítási hatásokat keltenek, sőt ha elég ügyesek vagyunk, a színek megválasztásával utánozhatjuk a különböző fényerőfokozattal működtethető drága terminálokat is.
Mi ezt kipróbáltuk, és úgy találtuk, hogy egy ilyen monitorral akkor érünk el optimális hatást, ha a háttér és a keret fekete, a mezők szürkék és a bebillentyűzött karakterek fehérek. Ez sokkal jobb megoldás, mint a monokromatikus számítógépeknél elterjedt rossz szokás, miszerint a változatlan képrészeket inverz ábrázolással (sötét betű világos alapon) írják a képernyőre, ami először is rosszul olvasható, másodszor nagyon kifárasztja a szemet, hiszen a normál íráshoz viszonyított kontrasztja nagyon nagy. Az inverz és a nagyon feltűnő ábrázolást csak akkor használjuk, ha különösen erős figyelmeztető hatást akarunk elérni, pl. hiba esetén.
Lehet, hogy az Olvasó némi ellentmondást fedez fel az általunk elmondottak és a professzionális programokban használt módszerek között. Nagyon sok forgalomban levő programcsomag képernyőmaszkja ugyanis hemzseg az inverz feliratoktól. De ha megfigyeljük, azok a programok, amelyek erősen igénybeveszik a gépkezelő szemét (pl. egy szövegszerkesztő program) sohasem dolgoznak inverz üzemmódban.
Az utóbbi időben egyre jobban terjed a teljes képernyő inverz megjelenítése, amelyben a képernyőháttér világos és az írás sötét. Elvileg ez jó optikai megoldás, azonban kivitelezéséhez különleges monitorokra van szükség, amelyek felbontóképessége valóban eléri egy írógéppel írt oldal finomságát. A hatás ahhoz hasonló, amit a vetítőgépek gyöngyvásznáról ismerünk. Egy normál tv-készülék azonban sohasem tudja ezt a képfelbontást és képtisztaságot nyújtani.
Szorítkozzunk most a C 64-esen elérhető lehetőségekre. Szín ajánlatunk és a hozzátartozó utasítások, melyeknek a program elején kell állniuk:

```
POKE 53280,0: rem fekete keret
POKE 53 281,0: rem fekete háttér
PRINT CHR$ (152) ; : rem szürke (2) betűk
PRINT CHR$ (5) ; : rem fehér betűk
```

ahol a fehér szín a tevékenységek, a szürke (2) pedig az állandó elemek, mint pl. a képmaszk kijelzésére szolgál. A következő fejezetben a képmaszk generálását egy konkrét példán keresztül mutatjuk be.
Foglaljuk össze mégegyszer az elmondottak lényegét: nem az a fontos, hogy egy kívülálló néző önkívületbe essen a képernyőn látható színgazdagság láttán, hanem az, hogy a későbbi gépkezelő a lehető legjobban tudjon dolgozni vele, és ne lásson maga előtt napokig rácsot egy órás munka után.

2.1.2 A beviteli mezők

Az általunk javasolt maszkfelépítés az adott 25 × 40-es képernyőfelbontásban, egy egysoros fejlécből, 23 beviteli sorból és egy lábjegyzet sorból áll.
Térjünk most vissza a korábbi fejezetekben bemutatott címfile példánkra, amelyben az első feladat a törzsadatok, azaz a név, cím stb. bevitele.

A fejléc tartalmazhatja pl. a következőket:

TÖRZSADAT-RÖGZÍTÉS FILE-NÉV

A beviteli mezőket – és ez minden beviteli módra érvényes – abban a logikai sorrendben, azaz abban a formában kell felépíteni, ahogyan a felhasználónak az adatok rendelkezésre állnak. Nálunk ez így néz ki:

Megszólítás
Név
Utca
Házszám
Postai irányítószám
Helység
1. megjegyzés
:
:
n. megjegyzés

A lábjegyzet így nézne ki:
BEVITEL VÉGE: F1 funkcióbillentyű a „Megszólítás" mezőbe.

A programot tehát úgy tervezzük, hogy a gépkezelő az adatbevitel végét az F1 funkcióbillentyű leütésével jelezheti. A Commodore 64-es gép szerencsére rendelkezik négy (Shift nélkül) funkcióbillentyűvel, amelyek általában az elektronikus adatfeldolgozásban nélkülözhetetlenek.
Hogy a funkcióbillentyűk valóban fontos szerepet töltenek be az adatbevitel során a kezelővezetés szempontjából, azt könnyen beláthatjuk.
Minden adatbeviteli lépésben felmerül a következő három lehetőség:

1. MINDEN ADAT HELYES – TÁROLNI KELL?
2. BEVITEL VÉGE (ez gyakran azt jelenti, hogy csak az aktuális résznek van vége és rendszerünk visszatér a főmenűhöz.)
3. Kényelmi okokból célszerű megengedni, hogy a gépkezelő egy mezőt visszaugorhasson a kurzorral, és kijavíthassa, ha hibát követett el.

A lehetőségek közötti választáshoz általában nem kell külön információt közölni a géppel, elegendő a választáshoz rendelt billentyűt leütni. Nagyon fontos, hogy az egyes lehetőségekhez rendelt billentyű necsak minden programrésznél, hanem lehetőleg minden programnál azonos legyen. Ez éppúgy hozzájárul ahhoz, hogy a program könnyen kezelhető legyen, mint az egységes maszkfelépítés.
A gépkezelő választását ellenőrző utasítás (alapulvéve a 2.1.4 alfejezet input rutinját) például a következő lehet:

```
100 F% = 1 : GOSUB 49000 : IF F$ = CHR$ (133) THEN END
```

Ezután a kis kitérő után térjünk vissza a képmaszkhoz. A fejezet végén közöljük a képmaszkot előállító rutin listáját. Biztosan sok tanulsággal jár, ha az Olvasó nem sajnálja a fáradtságot, és begépeli, kipróbálja ezt a szubrutint.

A szubrutin hívása előtt a következő változókról kell gondoskodni:

F% A mezők száma, mely 0 és 24 közé eshet. A 0. sor a fejléc, a 24. sor a lábjegyzet számára van fenntartva.
F$ A mező rögzített szövege (például: „Megszólítás")
F1% A mezők hossza. Maximális értékét a következő összefüggés adja meg: len F$+F1%≤37. Ha F1%=0, akkor ez a mező csupán kijelzésre szolgál és nincs szükség bevitelre.
FF% A mező típusa.
=0: alfanumerikus adat, hossza kisebb is lehet, mint F1%
=1: alfanumerikus adat, hossza kötött (F1%)
=2: numerikus adat, hossza kisebb is lehet, mint F1%
=3: numerikus adat, hossza kötött (F1%)

Példa:

```
10 FF% = 1 : F1% = 5 : FF% = 0 : F$ = "Megszólítás": GOSUB 51000
```

Ezzel tulajdonképpen az adatbevitel lényegi részét előkészítettük, felépítettük a képmaszkot és meghatároztuk az adatok típusát. Van még néhány további fontos lehetőség, amiről el kell döntenünk, hogy beépítjük a programba vagy nem. Megfontolhatjuk például, hogy lesz-e az adatmezők között olyan, amelynek tartalma az egyes címekben gyakran ismétlődik. Tegyük fel, hogy a program használója nagyon sok olyan ismerőse címét tartja nyilván, akik Budapesten laknak. Ha ezeket a címeket egymás után gépeli be, akkor minden esetben újra meg újra be kell írnia a HELYSÉG mezőbe a Budapest-et, holott ez fölösleges.
Célszerű tehát úgy megírni a programot, hogy egy-egy mezőn az előző adat ottmaradjon, és a gépkezelő a RETURN billentyű leütésével jelezhesse, hogy nem akar rajta változtatni. Erre a lehetőségre abban a programrészben is szükség van, amellyel az adatokat módosítjuk. A módosító programrész lemezről beolvas egy rekordot, kiírja a képernyőre és a gépkezelő úgy jelezheti, hogy egy mezőt (pl. az utcát) nem akarja módosítani, hogy megnyomja a RETURN billentyűt.
Azt a megoldást is választhatjuk, hogy mindig kiírjuk a képmaszkra azt az adatmezőt, ami előreláthatólag gyakran ismétlődik. Azt mondjuk pl., hogy a HELYSÉG adatmező alapértelmezése BUDAPEST. Az adatbevitel során a gépkezelő ha akarja, megváltoztatja, egyébként a RETURN billentyűvel nyugtázza.

```
20 F% = 1 : F$="BUDAPEST": GOSUB 50000
```

A következő kérdés, hogyan szervezzük meg a kilépést az adott programrészből ill. képmaszkból. A kilépést rábízhatjuk a gépkezelőre, de rábízhatjuk a programra is. Ez utóbbi főként olyan esetekben használatos, amikor a gépkezelőtől egyetlen kérdésre várunk választ, és amikor azt bebillentyűzte, a program visszatérhet a főmenűhöz, vagy áttérhet egy következő szubrutinba.
Hosszabb ideig tartó adatbevitelnél a gépkezelő jelzi, ha a munkát befejezte, és ki akar lépni az adatbeviteli munkafázisból.
Ezt a lehetőséget, és az üzenetküldés módját általában a legalsó sorban, a lábjegyzetben hozzuk a gépkezelő tudomására.

Például:

```
15 F%=24 : F$="PROGRAM VEGE: A MEGSZÓLITAS MEZŐBEN AZ F1 "GOSUB 51000
```

A következő oldal programlistájához még néhány megjegyzés: Az 51500-as sorban elhelyeztünk egy kis szubrutint, ami egy vagy több adatmezőt töröl a képmaszkról. A fejléc és a lábjegyzet természetesen érintetlenül marad. Az adatlap teljes törlésére minden kitöltött lap tárolása után szükség van, hiszen minden rekord bevitelét célszerű egy üres lapon kezdeni. Egy mező törlésére pedig akkor kerülhet sor, ha a gépkezelő hibás adatot ütött be. A törléshez csak a mező sorszámát kell megadni (F%). Ha F%=0, akkor a program az összes mezőt törli.

```
P2

50000 :
50001 REM **** ALAPERTELMEZES SZERINTI MEZOK BEVITELE ****
50002 :
50020 IFF%<1ORF%>23THENRETURN
50040 POKE214,F%:POKE211,F%(F%,2):SYS58732
50060 PRINTCHR$(5)LEFT$(F$(F%(F%,0));
50080 F%(F%,3)=1:RETURN
51000 :
51001 REM **** A KEPMASZK FELEPITESE ****
51002 :
51005 POKE53280,0:POKE53281,0
51010 IFF%<0ORF%>24THENRETURN
51020 CB$="                                                  "
51030 ONDF%GOTO51060
51040 DIMF%(24,3):DF%=1
51060 IFF%=0THENPRINTCHR$(19)CHR$(152)LEFT$(F$,40):RETURN
51070 IFF%<>24THEN51100
51080 POKE214,24:POKE211,0:SYS58732
51090 PRINTCHR$(152)LEFT$(F$,39);:RETURN
51100 F%(F%,0)=FL%:F%(F%,1)=FF%
51120 POKE214,F%:POKE211,0:SYS58732
51130 PRINTCHR$(152)F$;
51135 IFFL%=0THENRETURN
51137 PRINT" "CHR$(181);
51140 F%(F%,2)=PEEK(211)
51150 PRINTLEFT$(CB$,F1%)CHR$(182);
51160 F%(F%,3)=0
51180 RETURN
51500 :
51501 REM **** A MEZOK TORLESE ****
51502 :
51510 IFF%<0ORF%>24THENRETURN
51520 IFF%<>0THEN51600
51540 FORL1=1TO23
51560 F%=L1:GOSUB51600
51580 NEXTL1;RETURN
51600 POKE214,F%:POKE211,F%(F%,2):SYS58732
51620 PRINTLEFT$(CB$,F%(F%,0));
51640 F%(F%,3)=0:RETURN

READY.
```

A listán gyakran előfordul a SYS 58732 utasítás.
Ezzel az utasítással a kurzort a képernyő egy adott pozíciójára mozgatjuk, nevezetesen arra, amelynek koordinátái a 211-es és 214-es tárcímen találhatóak.

További kommentár helyett javasoljuk, hogy az Olvasó nézze át a programleírást:

50000–50080 Az alapértelmezés szerinti mezők kitöltéséhez a kurzort az F% (F%, 2) változóban megadott sorszámú mezőre mozgatjuk. Az F$-ban megadott konstans mezőérték hossza nem lehet nagyobb mint amit korábban megadtunk. Azt, hogy ez a mező foglalt, úgy jelezzük, hogy a mutatót a következő mezőre [F% (F%, 3)]-ra állítjuk. A mutató értékét az input rutin lekérdezi.

51000–51180 Az adatmező kezdetét a kurzor pozíciója, azaz a 211-es tárcím tartalma alapján határozzuk meg, és az F% (F%, 2) változóban tároljuk. Az adatmezőt feltöltjük szóköz karakterekkel. A mező foglaltsági jelzőjét nullázzuk [F% (F%, 3)].

51500–51640 A mező vagy mezők törléséhez az F%(F%,X)-ből kivesszük a mezővel kapcsolatos információt, a kurzort a mező elejére állítjuk és annyi üres karaktert írunk, amennyi a mező hossza.

A következő fejezetben bemutatjuk, hogy a rutinból hogyan veheti át a mezők tartalmát egy tetszőleges program.

2.1.3 A kurzor pozicionálása

Az adatbeviteli hibáknak beláthatatlan következményei lehetnek a feldolgozás eredményére. Ezért a programozónak minden igyekezetével azon kell lennie, hogy lehetetlenné tegye a beviteli hibákat. Olyan program persze nem létezik, amely minden létező hibát ki tudna védeni. (Meg kell jegyeznünk, hogy a hibák jellegét illetően a gépkezelők fantáziája, és képessége általában messze meghaladja azt, amit a programozó el tud képzelni.) Vannak azonban olyan módszerek, amelyek legalábbis csökkentik a hiba valószínűségét. Célszerű például optikailag is követni a begépelést, a leütések sorrendjének megfelelően balról jobbra ill. fentről lefelé, úgy, hogy közben a szemet mindig arra a helyre irányítjuk a kurzorral, ahol a gépelés tart.
A mintaprogram képernyőmaszkján külön megjelöljük az egyes mezők kezdetét és végét, ezzel teljesen kizárva azt, hogy valaki pl. a "Megszólítás" mezőbe helységnevet akarjon írni. A tényleges adatbevitelre a mező elején megjelenő kurzor szólítja fel a gépkezelőt. És ekkor kezdődik a legnehezebb programozói feladat. Semmiképpen nem engedhetjük meg, hogy a kurzor az adott mezőből bármilyen irányban kilépjen, pl. úgy, hogy a gépkezelő hosszabb adatot gépel be a megengedettnél. Tehát a kurzor mozgását állandóan ellenőrizni kell. Sajnos a kurzor pozicionálására a C 64-esen nem adottak a CBM 8032-es gépről ismert lehetőségek. Nincs lehetőség arra, hogy az INPUT és a GET utasításokban paraméterként megadjuk a kurzorpozíciót.
Nincs más megoldás, mint minden lépésben ellenőrizni, hogy a kurzor vízszintes ill. függőleges irányban a megengedett tartományban van-e.
Az operációs rendszer nulláslapján (0-tól 256-ig terjedő tárcímek) található két rendszerváltozó, amelyek az aktuális kurzorpozíció koordinátáit adják: a 211-es cím az aktuális oszlopot, a 214-es cím pedig az aktuális sort adja meg.
A két tárcímet csak akkor tudjuk minden leütésnél ellenőrizni, ha GET utasítással, karakterenként olvasunk be.
Egészen más a helyzet akkor, ha a gépkezelőnek kell eldöntenie, hogy melyik mezőt kívánja kitölteni, például, ha egy menű alapján különböző tevékenységek között választhat. Ilyenkor általában az egyes tevékenységekhez egyetlen számjegyet vagy betűt rendelünk, amelynek leütésével a gépkezelő jelzi a döntését. A maszkgenerátorba ezt a lehetőséget is beépítettük.

Így érthetővé válik az is, hogy miért volt szükség arra, hogy a nulla hosszúságú mezőnek is értelmet tulajdonítsunk.
Tegyük fel, hogy az egyes tevékenységeket (szubrutinokat) számjegyekkel jelöltük meg. A képernyőmaszkot a következő programsorokkal állíthatjuk elő:

```
P3

10 F%=0:F$="FOMENU":GOSUB51000
20 F%=2:F1%=1:FF%=3:F$="TORZSADATOK BEVITELE = 1":GOSUB51000
30 F%=4:F1%=0:F$="TORZSADATOK MODOSITASA = 2":GOSUB51000
40 F%=6:F$="NEV SZERINTI KERESES = 3":GOSUB51000
50 F%=24:F$="KEREM VALASSZON!":GOSUB51000

READY.
```

A fenti megoldás azért célszerű, mert így a gépkezelő egységes képmaszkokkal dolgozhat. Az adatbeviteli kép ugyanolyan szerkezetű, mint amit a megfelelő tevékenység kiválasztásához használunk.
Gyakori programozói hiba, hogy nemcsak különböző programrészekben, de még az egyes programrészeken belül is különböző elvárásokat támasztanak a gépkezelővel szemben. Az egyik beolvasáshoz INPUT utasítást használnak, és így az adat begépelése után a gépkezelőnek meg kell nyomnia a RETURN billentyűt, ugyanakkor később egy IGEN/NEM kérdésre adható válasz már csak egyetlen billentyű leütése RETURN nélkül.
Legjobb, ha minden adatbevitel után megköveteljük a RETURN billentyű leütését, ezzel ugyanis nyugtázhatja a gépkezelő a begépelt adat helyességét. Az INPUT utasítást azonban ha lehet, mindenképpen kerüljük el!
A gépkezelői tevékenységek irányítására használjuk a funkcióbillentyűket, amelyeket természetesen éppúgy, mint a RETURN billentyűt, nem kell nyugtáztatni.
Az elmondottakat ismét szemléltetjük a következő fejezet mintaprogramjában.

2.1.4 Beolvasás a billentyűzetről

Egyetlen beolvasási lépés három fázisból áll:

1. A billentyűzet lekérdezése
2. Az egyidejű vagy utólagos kijelzés a képernyőn
3. Az adat érvényességének folyamatos, vagy utólagos ellenőrzése.

A billentyűzet lekérdezésére két lehetőség kínálkozik, és mindkettőnek megvannak az előnyei és hátrányai egyaránt.

Az INPUT utasítás

Az INPUT utasítás előnye, hogy a képernyőn megjeleníti a villogó kurzort, ami a gépkezelőt felszólítja az adatbevitelre. A bevitel azonnal átkerül egy változóba, így nem kell bajlódnunk az egyes karakterek gyűjtögetésével, kezelésével.
Komoly hátrányt jelent azonban, ha az INPUT utasításba numerikus változót írtunk, és a program futása megszakad, ha a gépkezelő nem numerikus karaktert ír be. Ez különösen akkor kellemetlen, ha a gépkezelő és a programozó nem ugyanaz a személy, ugyanis a gépkezelő nem tudhatja, hogy a programot hogyan, honnan lehetne tovább folytatni.

A GET utasítás ebből a szempontból előnyösebb, mert minden karaktert kritika nélkül átvesz, de ezzel véget is ért az előnyeinek felsorolása.
Hátrányos, hogy a GET utasítás végrehajtása közben nem jelenik meg a kurzor. Tehát a beviteli felszólítást másként kell megvalósítani; vagy egy előző PRINT utasítással vagy a kurzor mesterséges bekapcsolásával. Ráadásul az egyes karakterekből össze kell állítani egy füzért úgy, hogy a gépkezelő bevitel közben egyetlen leütést sem korrigálhat. A hátrányok ellenére mi mégis a GET utasítás mellett döntöttünk és egy kis rutint, ami ezeket a problémákat – reméljük mindenki számára megnyugtatóan – megoldja, el is készítettünk. A második kérdés, hogy ti. mikor kell a bevitt adatokat a képernyőn kijelezni, tulajdonképpen csak a GET utasításnál vetődik fel. A GET utasítás ugyanis az INPUT utasítástól eltérően nem jelzi ki automatikusan a leütött karaktert, azaz tulajdonképpen "vak" bevitel történik. Egynél több karakter bevitele esetén semmiképpen nem kerülhetjük el, hogy a beolvasott karaktert azonnal a képernyőre kiírjuk, hiszen különben a gépkezelő nem tudná eldönteni, hogy helyes adatot ütött-e be vagy sem.
Az adatok érvényességének ellenőrzésére általában nagyon kevés figyelmet szentelnek. Pedig gyakran már a bevitel után hozzávetőlegesen azonnal megállapíthatjuk, hogy a beolvasott adat hossza vagy értéke egy előzetesen megadott tartományba esik, vagy nem. Így például valószínűtlen, hogy egy házszám öt számjegyű vagy egy élő személy születési dátuma 1880 előtti. Ha numerikus adatot olvasunk, minden lépésben ellenőrizhetjük, hogy a gépkezelő számjegyet ütött-e:

```
10 GET A$ : IF A$ = ""THEN 10
20 IF ASC (A$) < 48 OR ASC (A$) > 57 THEN 10
```

A fenti két sorral kizárjuk, hogy a gépkezelő a 0.....9 számjegyeken kívül mást is leüthessen. A feladat jellegétől függően persze numerikus adatbevitelnél meg kell engedni az előjelek és a tizedes pont leütését is.
A programírás egyik legnehezebb, legkevésbé látványos és ugyanakkor az egyik legfontosabb része az adatellenőrzés.
Feltételezzük, hogy minden Olvasót megörvendeztetünk azzal, hogy közöljük egy univerzális ellenőrző szubrutin listáját.
Az ellenőrzések beépítésével sok hiba kiszűrésére készítettük fel a programot, pl. meg tudja akadályozni, hogy a postai irányítószám mezőbe betűt vigyenek be, sőt kényszeríti a gépkezelőt arra is, hogy pontosan négy számjegyet írjon be.
Hogy ezek a védelmek mit jelentenek a program eredményessége szempontjából, azt feltehetően nem kell külön megvilágítani. A 2.1.2 pontban megmutattuk, hogyan kell az egyes beviteli mezőket definiálni, a következő rutin pedig arról gondoskodik, hogy ezeket az előírásokat a gépkezelő szigorúan be is tartsa.
A rutin kezelése nagyon egyszerű. A képmaszk felépítése után bármikor hívhatjuk GOSUB 49000 utasítással, előzetesen csak az F% értékét kell megadni, amely meghatározza a mező sorszámát.

```
100 F% = 2 : GOSUB 49000
```

A szubrutin a beolvasott eredményt az F$ változóban adja vissza. Ne feledkezzen meg az Olvasó arról, hogy numerikus mezőknél az F$-t át kell alakítania, pl. A=VAL (F$) utasítással, mielőtt műveletet végeznénk vele.

```
49000 :
49001 REM **** EGY MEZO BEOLVASASA ****
49002 :
49040 :
49060 F$="":IFF%(F%,0)=0THENRETURN
49080 IFF%<10RF%>23THENRETURN
49100 PRINTCHR$(5);
49120 ML=F%(F%,1)AND1
49140 CB$="                                                  "
49160 MN=1:IFF%(F%,1)AND2THENMN=0
49180 IL=F%(F%,0)
49200 CZ=F%:CS=F%(F%,2)
49220 POKE214,CZ:POKE211,CS:SYS58732
49240 F$="":CC=0:MS=0
49260 POKE204,0
49280 GETG$:IFG$=""THEN49280
49285 IFASC(G$)<>13THEN49300
49290 IFF%(F%,3)=1THENCC=IL
49295 GOTO49380
49300 PRINTLEFT$(CB$,IL);
49320 POKE211,CS:F%(F%,3)=0
49340 GOTO49380
49360 GETG$:IFG$=""THEN49360
49380 G=ASC(G$):IFG=13THENONML+1GOTO49640,49600
49390 IFG>132ANDG<141THENF$=G$:CC=0:GOTO49640
49400 IFG=29ANDPEEK(211)<=CS+IL-1THENGOSUB49820:GOTO49360
49420 IFG=157ANDPEEK(211)>CSTHENGOSUB49820:GOTO49360
49440 ONMNGOTO49520
49460 IF(G>47 AND G<58)ORG=45ORG=46THEN49540
49500 GOTO49360
49520 IFG<32OR(G>127ANDG<160)THEN49360
49540 IFCC<ILANDPEEK(211)<=CS+IL-1THENCC=CC+1
49560 IFPEEK(211)<CS+ILTHENGOSUB49820
49580 GOTO49360
49600 IFF%(F%,3)THEN49640
49620 IFPEEK(211)<>F%(F%,2)+ILORCC<ILTHEN49360
49640 POKE205,2
49660 IFPEEK(207)<>0THEN49660
49680 POKE204,1
49700 POKE211,CS:POKE214,CZ
49720 IFCC=0THENRETURN
49730 OPEN3,3
49740 GET#3,G$:IFG$=CHR$(13)THENF$=LEFT$(F$+CB$,IL):RETURN
49760 F$=F$+G$
49780 IFLEN(F$)<ILTHEN49740
49800 CLOSE3:RETURN
49820 POKE205,2
49840 IFPEEK(207)<>0THEN49840
49860 PRINTG$;:IFPEEK(211)>MSTHENMS=PEEK(211)
49880 RETURN

READY.
```

A szubrutin hívása előtt értéket kell adni a következő változóknak.

F%	a mező sorszáma
F% (F%, X)	a mező típusa
CC	az érvényes karakterek száma
IL	a mezőhosszúság

A rutin végén az F$ változó a beolvasott adatot tartalmazza.

49000–49260 A kurzor a beviteli mező első helyére áll és bekapcsol.

49280–49360 Egy karakter érkezik a billentyűzetről. Ha ez az első karakter volt, akkor a mező esetleges korábbi tartalma törlődik.

49380–49580 Ha a karakter RETURN (13), akkor a program a hosszúság ellenőrzésére ugrik. A program a kurzor jobbra/balra karaktereket közvetlenül végrehajtja, ha a célmező a beviteli területen belül van [ellenőrzés PEEK (211) utasítással]. Ha funkcióbillentyűt ütött le a gépkezelő, akkor visszatérünk a főprogramhoz. A többi karakter áthalad egy szűrőn (numerikus szűrő a 49560-as, vezérlőkarakter ellenőrzés a 49520-as sorban). Ezután ismét ellenőrizzük, hogy a kurzor még mindig a határokon belül található-e.

49600–49620 Az adatmező hosszának ellenőrzése. Ha az eddig beolvasott karakterek száma kisebb a mező hosszánál, vagy a mező nem volt előre lefoglalva, akkor ismét bevitel következik.

49640–49800 Ha nem vittünk be karaktert, akkor a kurzor kikapcsol és visszaugrik. Egyébként a program a bevitt karaktereket IL hosszúságban a képernyőről kiolvassa és az F$-ban átadja a felhasználónak.

49820–49880 Ez a kis szubrutin kiírja a beolvasott és a szűrűn áthaladt karaktereket. Kiíráshoz meg kell várni a kurzor sötét fázisát (207=0), mert egyébként a kurzor a régi helyén marad. Az időzítést BASIC nyelven nem túl könnyű meghatározni. Előfordulhat, hogy egy inverz mező marad a képernyőn, ami azonban az adatok érvényességét nem befolyásolja.

2.2 Az adatfeldolgozás

Tisztelt Olvasó! Tisztában vagyunk azzal, hogy az adatfeldolgozás a C 64-es programozásának legkényesebb területe. Számolunk azzal is, hogy akad majd Önök között olyan, aki hirtelen haragjában akár záptojással is megdobálna bennünket, ha a záptojással együtt mi is a közelében lennénk. Mégis vállaljuk a kockázatot, és nem rettenünk vissza e témakör tárgyalásától.
Kérjük, ne értsenek félre bennünket! Nem akarjuk eljátszani a tudós mester szerepét és nem akarjuk kioktatni az Olvasót.
Mégis talán tudunk néhány olyan újszerű megoldást mutatni ezen a területen, amire az Olvasó egyáltalán nem is gondolt, és ha így van, máris elértük célunkat.
Biztosan mindenkinek megvan az egyéni programozási stílusa, ez nem is baj és nem is kívánjuk, hogy bárki a kedvünkért a jól bevált módszerein változtasson.
Előfordulhat azonban mindenkivel, hogy programja túlságosan nagyra sikerült és gondjai támadnak a tárkapacitás körül. Mi a teendő ilyenkor? Elsőként erre a kérdésre keresünk választ.

2.2.1 A változók

A változók, főként a numerikus változók minden programozó játékszerei. Hogy is volna másként, hiszen remélhetőleg az Olvasó nem akar olyan programokat készíteni, mint amiről a 2.1 alfejezetben esett szó.
Gondolkodott-e már egyszer is arról, hogy mennyi helyet igényel egy változó a tárban?
A lebegőpontos változók 5 byte-ot foglalnak el, nem számítva az azonosítójukhoz szükséges 2 byte-ot, hiszen erre minden változótípusnak szüksége van.
Lebegőpontos alakban a gép minden számot kilenc értékes jeggyel és helypótló nullák nélkül, valamint egy kétjegyű hatványkitevővel ábrázol.
Az egésztípusú (%) változók (−32767-től +32767-ig) ugyanannyi helyet foglalnak el, mint a lebegőpontos változók, azaz összesen 7 byte-ot.
A füzérek ($) tárolásához is 7 byte-ra van szükség, 2 az azonosító, 1 a hossz, és ismét 2 egy mutató tárolására. Az utóbbi arra a tárcímre mutat, ahol a tényleges karakterek kezdődnek. Az adatterületen a füzér minden karaktere további 1 byte-ot foglal el.
Hogyan tárolja a gép az indexes változókat (tömböket), amelyeket DIM utasítással kell előkészíteni?
Az indexes változók helyigénye a következő:
A lebegőpontos tömbök elemenként 5 byte-ot, az egészértékű tömbök azonban elemenként csak 2 byte-ot foglalnak el.
A füzértömbök minden eleme 3 byte-ot foglal el, 1-et a hossz, 2-t pedig a mutató.
A tényleges karakterek tárolása itt is rugalmas.
Fontos azonban tudni, hogy a numerikus tömbök számára a DIM utasítás azonnal lefoglalja a szükséges tárterületet, a füzértömbök ezzel szemben csak az értékadást vagy az adatbevitelt követően bővülnek.
Milyen tanulságokat vonhat le a programozó a fentiekből? Például a következőket:
A numerikus változóknál nem érdekes, hogy lebegőpontosak vagy egész értékűek-e, mert helyigényük azonos. De az már egyáltalán nem mindegy, hogy egy tömb egész- vagy lebegőpontos típusú. Ha a feladat engedi, érdemes egész típusú tömbökkel dolgozni.

Az sem mindegy, hogy a lemezen hogyan tároljuk az adatokat. A 255-nél kisebb egész számokat például célszerű a lemezen karakteres formában tárolni. Az átalakítás módja a következő:

```
A$=CHR$ (A)
```

A visszaalakítás pedig:

```
A=ASC(A$)
```

Gyakran használjuk ezt a megoldást a QUISAM-file kezelésénél, ahol az egyik cél az adatok helytakarékos tárolása. A tárbeli feldolgozásra ez a módszer nem jó, ugyanis helyigénye nagyobb egy numerikus változóénál.

Hogyan takaríthatunk meg helyet a tömbkezelésnél?
Tegyük fel, hogy olyan algoritmussal dolgozunk, amihez feltétlenül deklarálnunk kell egy A(1000) típusú tömböt, holott az biztos, hogy az 1000 elem közül csak néhányra lesz szükségünk.
Dolgozzunk numerikus tömb helyett karakteres tömbbel, felhasználva az A$(X)=STR$ (A) utasítást, így rengeteg helyet takarítunk meg, és bármikor visszalakíthatjuk a füzért numerikus változóvá az A=VAL(A$(X)) utasítással.

Gondolt-e már az Olvasó valaha arra, hogy a lemezen tárolandó adatokat sűrítse?
Tetszőleges hosszúságú számokat sűríthetünk, ha nincsen bennük tizedes rész. Az eljárás nagyon egyszerű: felbontjuk a számot kétszámjegyű csoportokra, és minden kétjegyű szám helyett a megfelelő karaktert tároljuk a lemezen.

Egy további ötlet:
A tizedesvessző utáni számjegyek levágásához gyakran használják az INT függvényt. Ha biztosak vagyunk abban, hogy a kezelendő szám nem nagyobb, mint 32767, akkor az

```
A=INT(B)   utasítás helyett használjuk az
A%=B       utasítást.
```

Az értékadó utasítás végrehajtása gyorsabb, ugyanakkor szövege rövidebb, tehát helyet is megtakarítunk.
Reméljük, az Olvasó jónéhányat hasznosít majd az ötletek közül!

2.2.2 A számítási pontosság

Vannak kedvelt elrettentő példák, amiket bizonyos számítógépgyártmányok képviselői újra és újra felhoznak, ha a konkurrencia termékét kell szidalmazni.
Az egyik ilyen a

```
PRINT SQR(5)↑2
```

Mit gondol az Olvasó, mi lesz az eredmény?

```
5.00000001
```

A számítógép az utolsó helyiértéken hibát követett el.
Hasonló példa:

```
10 A=1
20 A=A-.1
30 PRINT A
40 GOTO 20
```

Természetesen azt várjuk, hogy a képernyőn sorban a 0,9; 0,8; 0,7... stb. számok jelennek majd meg. Sajnos nem ez a helyzet: 0,2 után a kapott értékek:

```
   .0999999998
-2.47382559E -10
```

A hibát az okozza, hogy az operációs rendszer véges sok bináris jeggyel, tehát korlátozott pontossággal tud csak számolni. A tárolt tíz decimális jegyből csak kilenc jelenik meg a képernyőn, és az utolsó helyiértéknél fellépő hiba bizony gyakran zavaró lehet. Különösen akkor, ha láncműveleteket végzünk, ilyenkor ugyanis a sorozatban keletkező hibák halmozódhatnak és a végeredményt már egészen durva hiba terhelheti.
A hibák halmozódásának megszüntetésére sajnos nem áll rendelkezésünkre bevált recept. Egy lehetséges megoldás a kerekítés lenne, ha tudnánk mikor kerekítsünk felfelé és mikor lefelé. Negatív és pozitív hiba egyaránt előfordulhat. Minden feladatban a számítások jellegétől kell függővé tenni, hogy milyen irányba kerekítsünk. A kerekítés mértéke is esetenként eltérő.
A kereskedelmi gyakorlatban általában két tizedesjegyre szoktunk kerekíteni, míg a tudományos számításokban (ha a fény hullámhosszával kell számolni) a kilencedik jegyre történő kerekítés is komoly hibát okozhat.
A pontosság és kerekítés kérdését tehát nem tudjuk végérvényesen megoldani. Mindössze annyit szerettünk volna elérni, hogy gondolatokat ébresszünk arra az esetre, amikor az Olvasó nem érti, hogyan kaphatott programjával teljesen váratlan számítási eredményeket.

2.2.3 A rendezés

A rendezésről nem egy fejezetet, hanem egy egész könyvet lehetne írni, ugyanis legalább annyi rendezési algoritmus létezik, ahány szoftverszerző.
A dolog lényege, hogy bizonyos adatokat növekvő vagy csökkenő (esetleg fele-fele arányban növekvő és csökkenő) sorba kell rendeznünk.
Valószínűnek tartjuk, hogy egyetlen olyan Olvasó sem akad, akinek ne lenne néhány ötlete a megoldáshoz.
Tulajdonképpen a nehézség nem magában a rendezési feladatban rejlik, hanem abban, hogy az eljárás lehetőleg gyors is legyen. A könyv szerzői nem szándékozzák szaporítani a rendezési algoritmusok amúgy is nagy számát. Ehelyett bemutatnak kettőt a leggyakrabban használt algoritmusok közül, remélve hogy ezzel gazdagítják az Olvasó programozói eszköztárát.

Az első eljárás nem egyéb, mint a manuális rendezés számítógépes megoldása. Próbáljunk rendbe rakni az asztalon egy pakli kártyát. A kártyákat nagyságuk szerint jobbra ill. balra rakosgatjuk anélkül, hogy közben bármilyen algoritmus járna az eszünkben.
Az eljárást *buborék-rendezés*nek is hívják, nem véletlenül, ugyanis a rendezés is valahogy ahhoz hasonlóan történik, mint amikor a légbuborék könnyűségénél fogva felszáll a folyadék felszínére. Gondoljunk arra, hogy az asztalon hogyan rendeződnek a kártyák. A kisebbek a jobb oldalon gyülekeznek, mindaddig, míg valamennyi a helyére nem kerül. Persze az ember tekintetét megragadja az ász, és így esetleg az egyetlen mozdulattal a helyére kerül. Ilyen frappáns cselekedetre a számítógép nem képes, neki egyenként kellene összehasonlítgatni minden kártyát a többivel, és ehhez már 7 kártya esetén is 2^7 számú összehasonlításra lenne szüksége.

A buborék-rendezés lényege a következő:
Két egymásbaágyazott ciklusban rendre összehasonlítjuk páronként az adatelemeket. Ha sorrendjük megfelelő, a helyükön hagyjuk, egyébként felcseréljük őket. A ciklusokat az L1 és L2 ciklusváltozók vezérlik, L2 a külső, L1 pedig a belső ciklus változója. Vesszük az adatok közül az elsőt, összehasonlítjuk a másodikkal, szükség esetén cserélünk, összehasonlítjuk a harmadikkal, negyedikkel stb., míg az adathalmaz végére nem értünk. Ekkor biztosak lehetünk abban, hogy a legnagyobb elem a sor végére került. Most megismételjük az egész eljárást elölről, de mivel tudjuk, hogy a legnagyobb elem a sor végére került, elég az összehasonlítást (cserét) az utolsó előtti elemig elvégezni.
Ha az Olvasó a leírás alapján nem érti világosan, hogy miről van szó, próbálja ki kártyákkal az asztalon végigjátszani az algoritmust.
Az eljáráshoz készített szubrutint alább közöljük. A szubrutin hívásakor a rendezendő adatokat az SD$(X) indexes változóban kell tárolni, és előzetesen deklarálni. Nem kell előre pontosan rögzíteni, hogy a tömb hány elemet tartalmaz, ezt a szubrutin saját maga megállapítja, ha utolsó elemként CHR$(255)-öt írunk bele. A szubrutint úgy írtuk meg, hogy automatikusan egy másik indexes változót, az SD%(X)-et elemzi és cserélget. Erre nagyon gyakran szükség van, hiszen ha például nem egyetlen adatmezőt, hanem egy teljes rekordot akarunk valamilyen mező (pl. név) szerint rendezni, célszerűtlen lenne a teljes rekordot minden lépésben cserélgetni. Ehelyett az SD%(X) indexes változóban megjegyezzük, hogy pl. az eredetileg első helyen álló rekord a rendezési szempont szerint hányadik helyre került, és rendezés után az SD%(X) tartalma alapján ebben a sorrendben vesszük elő az elemeket. Az elemek számát a szubrutin elején a CHR$(255) karakter figyelésével határozzuk meg, és az SN változóban tároljuk.
A szubrutin felépítése miatt az SD%(X) indexes változót akkor is deklarálnunk kell, ha tartalmát a későbbiek során nem használjuk fel. Ellenkező esetben a szubrutin BAD SUBSCRIPT ERROR hibaüzenettel megáll.
Ha semmiképpen sem akarunk élni ezzel a lehetőséggel, töröljük a rutinból a 60240-től 60280-ig terjedő sorokat.

```
P5

60000 :
60001 REM **** BUBOREKRENDEZES ****
60002 :
60060 SN=0:L1=255
60080 IFASC(SD$(SN))<>L1THENSN=SN+1:GOTO60080
60100 SN=SN-1
60120 FORL1=0TOSN-1:FORL2=L1+1TOSN
60140 IFSD$(L1)<SD$(L2)THEN60300
```

```
60180 SA$=SD$(L1)
60200 SD$(L1)=SD$(L2)
60220 SD$(L2)=SA$
60240 SA=SD%(L1)
60260 SD%(L2)=SD%(L1)
60280 SD%(L1)=SA
60300 NEXT:NEXT
60320 RETURN

READY.
```

Ha az adatokat nem növekvő, hanem csökkenő sorrendbe akarjuk rakni, a 60140-es sorban meg kell változtatni a relációjelet. A szubrutinnak 100 elem rendezéséhez kb. 175 s-ra van szüksége. A rendezési idő az elemek számával négyzetes arányban nő, tehát 1000 elemnél már 17500 s.

Kicsit bonyolultabb, de jóval gyorsabb rendezési eljárás az alábbi GYORSRENDEZŐ (QUICK SORT):
A GYORSRENDEZŐ 100 elemet 25 s alatt tud rendezni, hétszer olyan gyors, mint a buborék-rendezés, tehát valóban illik rá a „gyors" jelző. Az időszükséglet az elemek számával ugyan a lineárisnál kicsit erősebben, de semmiképpen nem négyzetesen nő. Pedig a program alig hosszabb, mint a buborék-rendezés programja.

Hogyan működik ez az algoritmus?
Vegyük elő ismét a kártyákat.
Válasszuk ki a középső kártyát, hasonlítsuk össze ezt páronként a tőle balra állókkal és jegyezzük meg, ha találtunk nála kisebbet. Most hasonlítsuk össze a középső kártyát a tőle jobbra esők mindegyikével, és ha találtunk nála nagyobbat, azt cseréljük fel azzal, amit a baloldaliak között az előbb kiszemeltünk. Ismételjük ezt a lépést mindaddig, amíg az összehasonlítással nem érkezünk középre. Ha most a kártyákat megnézzük, azt látjuk, hogy a középső kártyától balra a nála nagyobbak, jobbra a kisebb értékűek találhatók. Ez azt jelenti, hogy egy durva előrendezéssel elkészültünk.
Most felezzük meg a kártyacsomag jobb oldali felét, ismételjük meg az eljárást a jobb szélső és a jobb oldali második negyeddel. Ily módon egyre kisebb darabokat kell kezelni, végül már csak két kártya marad. Ekkor az utoljára kezelt jobb külső egység kártyái már rendezettek, vehetjük a következő egységet, egészen addig, míg el nem érjük a halmaz közepét. A tőle jobbra eső kártyák sorrendje megfelelő. Áttérhetünk a kártyacsomag baloldali felére. Az eljárást egészen addig kell folytatni, amíg minden kis egységgel nem végeztünk (amíg két kártya nem alkot egy egységet).
Annak ellenére, hogy az algoritmus első látásra bonyolultnak tűnik, valójában nem az, és programtechnikai megoldása sem okoz különösebb nehézséget.
A rutin paraméterei majdnem azonosak az előző rendező rutin paramétereivel. Vizsgáljuk meg az egyes változók jelentését:

SD$	A középső elem
SV$	A felcseréléshez szükséges átmeneti tároló
S1(X)	A még feldolgozandó egység baloldala
S2(X)	A feldolgozandó egység jobboldala
S1	Az aktuális rész bal szélső eleme
S2	Az aktuális rész jobb szélső eleme

SP Mutató a szélső párra. Miközben az adathalmaz szélétől a közepe felé haladunk, egy részhalmaz kezelése után arról a területről töltünk be egy szélső párt, ahol az elemek már rendezetten állnak.

L1, L2 A középső elemmel összehasonlított elemek indexei.

Az alábbiakban közöljük a GYORSRENDEZŐ szubrutin listáját.

```
P6

61000 :
61001 REM ***** QUICKSORT *****
61002 :
61020 ONSF%GOTO61060
61040 DIMS1(15),S2(15):SF%=1
61060 SN=0:L1=255
61080 IFASC(SD$(SN))<>L1THENSN=SN+1:GOTO61080
61100 SN=SN-1:SP=0
61120 S1(0)=0:S2(0)=SN
61140 S1=S1(SP):S2=S2(SP)
61160 SP=SP-1
61180 L1=S1:L2=S2
61200 SD$=SD$((S1+S2)/2)
61220 IFSD$(L1)<SD$ANDL1<S2THENL1=L1+1:GOTO61220
61240 IFSD$(L2)>SD$ANDL2>S1THENL2=L2-1:GOTO61240
61260 IFL1>L2THEN61340
61280 SV=SD%(L1):SD%(L1)=SD%(L2):SD%(L2)=SV
61300 SV$=SD$(L1):SD$(L1)=SD$(L2):SD$(L2)=SV$
61320 L1=L1+1:L2=L2-1
61340 IFL1<=L2THEN61220
61360 IFS2-L1<=L2-S1THEN61420
61380 IFS1<L2THENSP=SP+1:S1(SP)=L1:S2(SP)=L2
61400 S1=L1:GOTO61460
61420 IFL1<S2THENSP=SP+1:S1(SP)=L1:S2(SP)=L2
61440 S2=L2
61460 IFS2>S1THEN61180
61480 IFSP>-1THEN61140
61500 RETURN

READY.
```

A bemutatott algoritmusok közül az első az egyik leglassúbb, a második pedig az egyik leggyorsabb algoritmus.
Az összehasonlítást átlagok alapján végeztük, hiszen egyetlen algoritmus végrehajtási ideje sem független az adathalmaz véletlen rendezettségi fokától.
Ahányszor lefuttatjuk a szubrutinokat, mindannyiszor eltérő végrehajtási időket kapunk, ezek átlagát vettük figyelembe az összehasonlításnál.
A buborék-rendezés végrehajtási idejét jelentősen lecsökkenthetjük úgy, hogy figyeljük a belső ciklusban előfordult cserék számát. Ha ugyanis a belső ciklusban már egyetlen csere sem történt, fölösleges tovább folytatni az eljárást, mert egészen biztos, hogy minden elem a helyére került.
Kis elemszámnál (max. 10 elemig) nyugodtan használhatjuk a buborék-rendezést, nagyobb tömbök rendezésére nyilván a második algoritmus előnyösebb.

Programozástechnikailag szorosan kapcsolódik az adathalmazok rendezési kérdéseihez az adathalmazban való keresési probléma. A keresésnél legalább olyan fontos paraméter az idő, mint a rendezésnél. A keresési algoritmusok is követik a rendezési algoritmusok logikáját: kereshetünk lineárisan (buborék-rendezés) vagy úgy, hogy az adathalmazt részekre bontjuk, és először a részhalmazokat vizsgáljuk meg (gyorsrendezés).
Ráadásul a keresési idő most is az adathalmaz rendezettségi fokának függvénye. Ha az elemek rendezetten követik egymást, a keresés nagyon egyszerűvé válik, de itt is gyorsabb a felező eljárás a lineáris módszernél.
Összehasonlítjuk a keresett elemet az adathalmaz középső elemével. Ha a keresett elem kisebb, és az adathalmaz növekvő sorba van rendezve, biztos, hogy a keresett elem az adathalmaz alsó felében van. Ezt ismét megfelezzük, és a középső elemet hasonlítjuk a keresettel. Az eljárást addig folytatjuk, amíg az aktuális részhalmaz egyetlen elemből nem áll, és akkor készen vagyunk.
A keresési algoritmushoz nem készítettünk programot, nem akarjuk megsérteni az Olvasót.

2.3 Az adatok kiírása

A végletekig leegyszerűsítve, a számítógépes feldolgozás három lépésből áll: valami bemegy a gépbe, ott benn valami történik, végül valami kijön a gépből. Természetesen számunkra az a legfontosabb „ami kijön", vagyis az egész feldolgozás eredménye. Hogy az eredményeket milyen formában és hová kellene kiírni, arról nagyon sok programozó munka közben hajlandó egészen megfeledkezni. Beletemetkezik a programozás közben támadt jobbnál jobb ötletek megvalósításába, nagy megelégedéssel próbálgatja a már működőképes algoritmusokat, de az nem jut eszébe, hogy a feldolgozás eredményeit valamikor a kiíratáshoz is elő kellene készíteni. A program befejezése felé közeledve villámcsapásként éri a felismerés, hogy a kényelmes adatkiíráshoz egy-két paramétert félúton érdemes lett volna felvenni, hogy pl. egy kiírandó füzér hossza, amire most nagy szüksége lenne, már rég a múlté, az eredeti változót ugyanis felhasználta egy későbbi programlépésben és így tovább, és így tovább.
A programírásnak ezt a keserű felismerésektől gazdag fázisát Olvasóink közül biztosan sokan ismerik.
Mit lehet tenni annak érdekében, hogy az ilyen pillanatok kimaradjanak programozói életünkből?
Például a következőket:
Minden adatról, első előfordulásának pillanatában meg kell vizsgálni, hogy szükségünk lesz-e rá a kiírás során és ha igen, akkor milyen formában.
Előre meg kell gondolnunk, hogy az adatokat vagy eredményeket melyik egységre fogjuk kiírni: képernyőre, lemezre vagy nyomtatóra. A képernyőre történő kiírással nem kívánunk részletesen foglalkozni, egyrészt mert a képernyő elsősorban az adatbevitelnél játszik fontos szerepet, a beviteli képmaszk megszerkesztéséről pedig már bőven esett szó.
A következő fejezetekben a lemezes kiírással ill. a nyomtatással foglalkozunk.

2.3.1 Az adatok kiírása lemezre

Aki először találkozik a lemezegységgel, többnyire ijedtében lapozgatja a lemezegység kézikönyvét. A lemezkezelés utasításai nem egyszerű PRINT és INPUT utasítások, mint amit a nyomtatásnál vagy a képernyőre történő kiíratásnál használtunk, hanem rögtön az OPEN utasítással kezdődnek.
A nem hivatásos programozók számára az OPEN utasítás jóval idegenebb, mint bármelyik BASIC utasítás. A lemezműveletek közül a legegyszerűbbek a programok tárolási és betöltési műveletei. Ha valaki ezeket már rutinszerűen végzi, áttérhet a következő lépésre, a soros file-ok programozására. Ahol a soros file-on túli lemezműveletek leírása kezdődik a kézikönyvekben, ott a legtöbb műkedvelő könyve becsukódik. Néhány igazán ínyenc szakmabelin kívül ki szeret bíbelődni az olyan körülményes eljárásokkal, mint pl. a közvetlen elérésű utasítások, a Block-Read, Block-Write stb?
Ennek ellenére mi azt valljuk, hogy rosszul jár, aki egyáltalán nem néz utána a lemezegység programozási lehetőségeinek. Van ugyanis néhány viszonylag egyszerű dolog, mint pl. a relatív file, amiről a kézikönyvben alig esik szó, holott nem is túlságosan bonyolult és igen jó adattárolási lehetőség.
Mielőtt a relatív file tárgyalására rátérnénk, bemelegítésképpen szánjunk még néhány gondolatot a legegyszerűbb adatkezelési módszerre, a soros file kezelésére.

2.3.1.1 A SEQ

A soros (SEQuential) file meglepő módon a leggyakrabban használt file-típus. Olyan feladatok megoldásában is alkalmazzák, amikor egyáltalán nem praktikus, ugyanis gyakran kell a tárolt adatok között keresni. Ennek általában az az oka, hogy a programozó eszköztárában a soros file-nak nincsenek valódi alternatívái.
A soros adattárolás lényege már a nevében is benne van. A beérkező adatok PRINT# utasítással egymás után sorban felíródnak a lemezre (előtte a pufferbe).
Visszaolvasni pontosan ugyanabban a sorrendben tudjuk, ahogy kiírtuk őket.
Ez tökéletesen rendben van – gondolja az Olvasó –, de mi ebben a célszerűtlen?
Képzelje el az Olvasó, hogy egy 100 m hosszú zsinórt fel kell tekernie. Közben eszébe jut, hogy egy darabot pirosra fest belőle. Ezt meg is teszi, majd az egészet gondosan feltekercseli. Egy héttel később kizárólag a piros darabra van szüksége. Mivel az a gombolyag közepén van, nem fér máshogyan hozzá, csak úgy, ha az előtte lévő összes zsinórt letekeri. Praktikus? Pontosan ugyanez a helyzet a SEQ típusú file-lal és pontosan itt van a bökkenő. A gombolyag csak akkor használható, ha a kötélre, mint egészre van szükségünk.

Egyáltalán mire jó akkor ez a file-típus?
Nagy mennyiségű adatok gyors tárolására szolgál, olyan adatok tárolására, amelyekre a későbbiekben teljes egészében és változatlan sorrendben lesz szükségünk.
Az egyik alkalmazási lehetőség pl. a változók tartalmának lemezre mentése.
A professzionális programokban is alkalmazott gyakorlat az értékek folyamatos írása a lemezre. Gondoljunk csak a kasszára. Minden bevételt sorban beütnek a gépbe. Esténként csak a file tartalmát kell egymás után beolvasni és összeadni ahhoz, hogy a napi forgalmat megkapják.
Hogy a folyamatos írásnál maradjunk:
Elméletileg előfordulhat az az eset, hogy a tárolás nem folyamatos, egy-egy adat vagy adatok tárolása után eltelik bizonyos idő. Célszerű lenne ilyen esetekben a biztonság kedvéért a file-t minden hozzáférés után újra lezárni. Igen ám, de hogyan lehet új adatokat hozzátenni a meglévőkhöz?
A kézikönyv csak két soros hozzáférési módot ismertet, nevezetesen:

```
OPEN lfn, "NEV,S,W" megnyitás írásra
OPEN lfn, "NEV,S,R" megnyitás olvasásra
```

Próbáljon meg egy már meglévő file-t megnyitni olvasásra, a lemezegységen azonnal villogni kezd a piros lámpa és megjelenik a FILE EXISTS ERROR hibaüzenet.
Ravasz programozók azt csinálják, hogy a régi adatokat beolvassák a tárba, felírják egy másik file-ba és amikor a régi adatok elfogytak, folytatják az írást az új adatokkal. A következő napon ugyanaz a játék ismétlődik.
Jóval egyszerűbben is megoldható a file bővítése:

```
OPEN lfn, "NEV,S,A"
```

ahol az A hozzáfűzést (APPEND) jelöl, ezzel a hozzáféréssel új adatokat fűzhetünk a már meglévőkhöz.
Hogy ez miért nem szerepel a kézikönyvben? Azt mi sem tudjuk.

Foglaljuk össze az elmondottakat:
A soros file a későbbiekben teljes egészében ismét szükségessé váló adatok gyors tárolására alkalmas.
Nem alkalmas azonban olyan adatok tárolására, amelyeket a későbbiekben nem meghatározható sorrendben kell visszaolvasni.

2.3.1.2 A REL

A cím egy új fogalmat takar. A REL a relatív file rövidítése. Milyen adattárolásról van itt szó? Térjünk vissza (ami ugyan kissé sántít, de a célnak megfelel) a zsinóros példánkra.
Van egy sor számozott fiókunk, melyek mindegyikébe pontosan 1 m zsinór fér. Mivel gyakran van szükségünk különböző színű zsinórokra, ezek közül néhányat előre elkészítettünk és külön fiókba helyeztünk.
Ahhoz, hogy ne kelljen minden esetben megnéznünk a színt (a fiókok zárva vannak, először ki kellene nyitnunk őket), készítettünk egy jegyzéket, amiből kiderül a fiókszám és a zsinórszín közötti összefüggés. Ha a jegyzékbe bepillantunk, máris egy mozdulattal elővehetjük a kívánt színű zsinórt.
A példából két fontos dolog derül ki:
1) Teljesen mindegy, hogy mi és milyen sorrendben van a fiókban (lehetnek csavarok és radírgumik is), csak az a fontos, hogy a kezünkben legyen az elrendezésnek megfelelő lista. Gondoljunk a könyvtárosra, aki anélkül, hogy odapillantana, már nyúl is a megfelelő könyvért.
2) Minden fiók azonos méretű, vagy három radírgumi vagy 100 csavar fér bele.

Térjünk most vissza az adatfeldolgozáshoz és a radírgumik ill. csavarok helyett gondolkodjunk pl. különböző hosszúságú füzérekben.
Soros file esetében a név már utal az adattárolás módjára. De mire utal a „relatív" jelző? A REL file azonos hosszúságú tárterületek sokaságából áll, amelyeket a file létrehozásakor előre lefoglalunk, hogy a használó szabadon hozzáférhessen az egyik ilyen területhez – függetlenül attól, hogy olvasásról vagy írásról van-e szó – anélkül, hogy tudnia kellene, hogy ez a hely a lemezen fizikailag hol található. A lemezegység operációs rendszere, a DOS ún. rekordmutatókat képez. Ha közöljük a lemezegységgel, hogy a tizedik rekordot szeretnénk elérni, akkor ezt a sorszámot a DOS a file-kezdettől számított tizedik helyként – tehát relatív sorszámként – értelmezi. A mutató rááll a kiválasztott rekordra és attól függően, hogy a következő utasítás PRINT# vagy INPUT#, a számítógép felírja vagy elolvassa a szóban forgó adatterületet.
A REL file egyidejűleg mindkét hozzáférési mód számára nyitott. Az OPEN utasításban nem kell külön meghatározni, hogy írni akarunk a file-ba vagy olvasni akarunk belőle.
Most térjünk át a részletekre. Hogyan közölhetjük a lemezegységgel hogy hány és milyen hosszúságú rekord számára szeretnénk helyet foglalni? A kézikönyvből ez nem derül ki. Valójában a gondot nem is a lemezegység, hanem a C 64-es operációs rendszere okozza, amely nem rendelkezik a „nagy" Commodore számítógépek BASIC 4.0 programnyelvében adott kényelmes lemezkezelő utasításokkal.
A hiányzó utasításokat a C 64-esen „gyalogszerrel" kell előállítanunk. Lássunk példaként egy kis rutint, majd vizsgáljuk meg közelebbről a benne szereplő utasításokat.

```
P7

63800 :
63801 REM **** A RELATIV FILE FELEPITESE ****
63802 :
63810 IFRL%<2 OR RL%>254 OR R>65535 THENF1$="99":RETURN
63820 OPEN6,8,6"0:"+DN$+",L,"+CHR$(RL%)
63840 RH%=R/256:RL%=R-256*RH%
63860 PRINT#15,"P"CHR$(6)CHR$(RL%)CHR$(RH%)CHR$(1)
63880 PRINT#6,CHR$(255)
63900 INPUT#15,F1$,F2$,F3$
63920 CLOSE6:RZ$="":RETURN

READY.
```

A rutin feladata, hogy a lemezen lefoglaljon annyi megadott hosszúságú adatterületet (rekordot), amennyit előírtunk.
A rekordok számát tulajdonképpen nem kellene előre megadni, hiszen a file feldolgozás közben folyamatosan bővülhet.
Azonban ha pl. az 1000. rekordot akarjuk felírni a lemezre menet közben – ami az előre tervezett legnagyobb sorszámú rekord –, akkor a DOS a file-t felduzzasztja, az addig még nem foglalt rekordokat előkészíti, egészen a 999. rekordig. Ennek a következő feldolgozás szempontjából az az előnye, hogy a feldolgozás gyorsabban megy, mivel az új rekordokat már nem kell előkészíteni, mert – mégha üresen is – rendelkezésre állnak. Ha a feldolgozás menetét meg akarjuk gyorsítani, célszerű a file első megnyitásakor a teljes adatterületet előkészíteni. A rekordok számával szemben az egyes rekordok hosszát mindig előre kell rögzíteni. A DOS-nak erre feltétlenül szüksége van ahhoz, hogy megfelelő méretű területet foglaljon le (ez hozzávetőleg: rekordhossz × a legnagyobb rekord sorszáma). A fenti szubrutin, ami előkészíti a megfelelő területet, a következő változókat használja:

DN$ A file neve
RL% Egy rekord hosszúsága. Az RL% értéke legalább 2, de legfeljebb 254 lehet. Ellenkező esetben a lemezegység hibát jelez. RL% további felhasználását lásd RH%-nál.
R A legnagyobb rekordsorszám, amely meghatározza az előkészítendő adatterület méretét. R értéke 1 és 65535 közé kell hogy essen. Ha feldolgozás közben az először megadott értéknél mégis nagyobb sorszámú rekordra hivatkozunk, akkor a DOS a file-t automatikusan megnöveli, ha a lemezen van elegendő hely. A 65535-öt azonban semmiképpen sem léphetjük túl.
RH% A rekordszám (R) értékét két byte-ra bontjuk, amiből RH% az R szám 256-tal történő osztásának hányadosát az RL% pedig az osztás maradékát tartalmazza.
F1$ Hiba (ha értéke 0, nem történik hiba)
F2$ A hibaüzenet szövege (pl. FILE NOT FOUND)
F3$ A hibaüzenet bővítése (a file-ok törlésénél például a törölt file-ok száma)

A program működésének magyarázata:

63810 Az RL% és az R paraméterek érvényességi vizsgálata. Ha egy érték érvénytelen, akkor az F1$-ba egy hibaüzenet kerül. Minden hívás után le kell kérdezni az F1$ értékét, hogy meggyőződjünk a művelet hibátlan kimeneteléről.
63820 A 6-os másodlagos címmel az adatcsatornán keresztül átadjuk a meghajtószámot (itt nulla), a file nevét, az L karaktert a file-típus azonosítására, végül a kívánt rekordhosszúságot.

63840 Az R változó értéket két byte-ra bontjuk. Az INT függvény helyett egész típusú változóval dolgozunk.
63860 Előkészítjük a file-t. A parancscsatornán keresztül átadjuk a pozicionálás "p" azonosító jelét, az adatcsatorna számát, a két részre bontott rekordszámot, végül a rekordon belül a kívánt pozíciót. A bemutatott szubrutinokban mi a mutatót mindig 1-re, azaz a rekord első karakterére állítjuk.
63880 A file rekordjait csak úgy tudjuk előkészíteni, ha legalább egy karaktert mindegyikbe beleírunk. Erre a célra a π karaktert CHR$(255) használjuk. A későbbi olvasásnál a π azt jelzi, hogy az adott rekordba tényleges adatot még nem írtunk. (Ha a DOS a feldolgozás során üres rekordot talál, szintén a CHR$(255) karaktert írja bele.)
63900 A hibacsatorna leolvasásával ellenőrizzük, hogy a lemezművelet hibamentesen zajlott-e le.
63920 A csatornákat lezárjuk.

Figyelem:

Abban a programban, amelyből hívjuk a rutint, OPEN 15,8,15 utasítással meg kell nyitni, majd a program végén CLOSE 15 utasítással le kell zárni a parancscsatornát.

Most már elő van készítve a file. Az Olvasó biztosan kíváncsian várja, hogyan töltjük meg adatokkal. Ahogyan a nyomtatásnál, itt is meg kell adnunk a kiírási pozíciót. A relatív file esetén a pozicionálás a rekord sorszámának előzetes megadását jelenti.
Mielőtt önálló programot írna az Olvasó, nézze át alaposan a következő szubrutint, és olvassa el a hozzáfűzött magyarázatokat.

```
PB

63000 :
63001 REM **** A RELATIV FILE IRASA ****
63002 :
63010 IFRZ$=DN$THEN63040
63020 CLOSE6
63030 OPEN6,8,6,"0:"+DN$:RZ$=DN$
63040 RH%=R/256:RL%=R-256*RH%
63060 PRINT#15,"P"CHR$(6)CHR$(RL%)CHR$(RH%)CHR%(1)
63080 PRINT#6,R$
63100 INPUT#15,F1$,F2$,F3$
63120 RETURN

READY.
```

Az előző szubrutinban nem szerepelt az R$ változó. Az R$ változó tartalmazza a lemezre írandó rekordot.

63010 Az időmegtakarítás érdekében az OPEN-t átugorjuk, ha utoljára ugyanezzel a file-lal dolgoztunk.
63020 Megnyitjuk az adatcsatornát. Ha a file már létezik, az OPEN utasításban nem kell megadni az "L" file-típus paramétert és a rekordhosszt, mivel a DOS ezt az információt a tartalomjegyzékből megszerezheti.
63040 Az R változó felbontása
63060 Pozicionálás
63080 Az adatok felírása a file-ba
63100 A hibacsatorna lekérdezése.

A file-t előkészítettük, és beleírtunk egy rekordot, most már csak a visszaolvasás van hátra. Az olvasási utasítás előtt meg kell adni a keresett rekord sorszámát.
Az adatok visszaolvasását a következő szubrutinnal végezhetjük el:

```
F9

63500 :
63501 REM **** A RELATIV FILE OLVASASA ****
63502 :
63510 IFRZ$=DN$THEN63540
63520 CLOSE6
63530 OPEN6,8,6,"0:"+DN$:RZ$=DN$
63540 RH%=R/256:RL%=R-256*RH%:R$=""
63560 PRINT#15,"P"CHR$(6)CHR$(RL%)CHR$(RH%)CHR$(1)
63580 INPUT#15,F$1,F$2,F$3:IFF$1<>"00"THEN63720
63590 IFRC%=255THEN63740
63600 FORL1=1TORC%
63620 GET#6,RI$:IFRI$=""THENRI$=CHR$(0)
63630 INPUT#15,F$1,F$2,F$3:IFF$1<>"00"THENL1=RC%:GOTO63680
63660 R$=R$+RI$
63680 NEXTL1
63700 IFLEFT$(R$,1)=CHR$(255)THENF1$="50"
63720 RETURN
63740 GET#6,RI$:IFRI$=""THENRI$=CHR$(0)
63760 INPUT#15,F$1,F$2,F$3:IFF$1<>"00"THEN63700
63770 IFRI$=CHR$(13)THEN63700
63780 R$=R$+RI$:GOTO63740

READY.
```

Néhány új változó:

L1 Ciklusváltozó
RI$ Átmeneti tároló az R$ változó töltéséhez
RC% Ez a változó megadja, hogy hány byte-ot kívánunk az adott rekordból beolvasni. Ha az egész rekordot be akarjuk olvasni, akkor az RC% értéke legyen 255. A rutin amúgy is csak RETURN-ig [CHR$(13)] olvas, ami a tömb logikai végét jelzi.

A szubrutin dokumentációja:

63520–63580 A hibacsatorna lekérdezése. Ha F1$ = 56, akkor olyan rekordra pozicionáltunk, amely nincs előkészítve [nincs benne CHR$(255)].
63600–63680 A byte-okat egyenként beolvassuk és hozzáadjuk az R$ változóhoz. Ez az eljárás ugyan valamivel hosszabb ideig tart, mintha INPUT# utasítással olvasnánk be, de az az előnye, hogy nem kell tekintettel lennie az adattípusra (mint a 2.1.4-ben) és ezen kívül olyan különleges karakterek is beolvashatók, amelyeknél minden INPUT megszakítaná a folyamatot, például a tizedesvessző és CHR$(0). Az R$-t addig töltjük, amíg az RC%-kal adott felső határt el nem értük. Természetesen a program minden byte beolvasása után lekérdezi a hibacsatornát.
63700 Mint a fentiekben már említettük, egy előkészített, de adatokat még nem tartalmazó rekord első karaktere CHR$(255). Ilyen esetben mesterségesen hibaüzenetet generálunk az F1$ változóban (BLOCK NOT AVAILABLE).
63740–63780 Ha RC% = 255, csak addig olvasunk, amíg a RETURN karakterhez nem érünk.

A bemutatott három rutinból bármilyen relatív file-kezelő programot felépíthetünk. Ugye nem is olyan bonyolult, vagy mégis? Mint mindenütt, itt is érvényes: a tapasztalat többet ér az elmélkedésnél.
Végül foglaljuk össze még egyszer a SEQ és a REL file-típussal kapcsolatos alapgondolatokat a címnyilvántartási és a pénztárkezelési feladatok összehasonlításával.
Az első példában a címek tárolása után egy meghatározott névhez tartozó adatokat (pl. cím stb.) kellett visszakeresnünk. Ezeket az adatokat nem volna célszerű soros file-ba helyezni, mivel a keresés rendkívül sok időt venne igénybe, hiszen közben sok fölösleges rekordot is be kellene olvasni. Sokkal praktikusabb a címeket relatív file-ban tárolni, hiszen ha készítünk egy jegyzéket a címek tárolási sorrendjéről, akkor a sorszám alapján bármelyiket azonnal visszakereshetjük.
Másfelől értelmetlen volna folyamatos feldolgozású adatokat relatív file-ba helyezni, mivel a relatív file helyszükséglete azonos adatmennyiség esetén jóval nagyobb, mint amennyit a soros tárolás igényel.
A következő fejezetben egy olyan file-struktúrát mutatunk be az Olvasónak, ami mind a két előnyt egyesíti, azonban sajnos nem valósítható meg a VC–1541-es típusú lemezegységen.

2.3.1.3 Az ISAM

Sajnos az ISAM file kezelése nincs beépítve a VC–1541-es operációs rendszerébe, és bár programozását meg lehet oldani a C 64-esen, ez korántsem olyan kényelmes, mint a relatív file programozása. Mégis miért foglalkozunk ezzel a file-típussal?
Egyrészt, mert az a véleményünk, hogy a magasabb szintű file-szervezési módszerek megismerése semmiképpen nem árt azoknak, akik szeretnének elmélyülni a programozásban.
Másrészt, mert az új ismeretek olyan kísérletekre ösztönzik az Olvasót, amelynek végeredménye adott esetben egy saját, kényelmes file-szervezési eljárás kidolgozása lehet.
Az ISAM az Indexed Sequential Access Method angol szavak rövidítése. A megnevezés magyar jelentése: indexelt soros hozzáférési mód.
Mit jelentenek ezek a szavak?
Idézzük fel egy pillanatra a relatív file felépítését. A relatív file azonos hosszúságú rekordokból áll, melyek mindegyikét egy kód segítségével érhetjük el, ez az ún. rekordszám, vagy elérési kulcs. A rekord kulcsa és a rekord tartalma között semmiféle logikai összefüggés nincs, a kulcsból nem következtethetünk a rekord tartalmára és megfordítva. A kapcsolatot a rekordok és a kulcsok között egy tartalomjegyzékkel teremthetjük meg, ha szükségünk van rá.
A relatív file-ban a rekordok növekvő kulcs szerint követik egymást. Ebből következik, hogy a relatív file-t elejétől kezdve sorosan is végigolvashatjuk.
Az ISAM file megnevezésében a „soros" jelző pontosan erre utal. Bennünket persze most elsősorban nem az érdekel, hogy az ISAM file-típus miben hasonlít az eddig megismert file-típusokra, hanem éppen az, hogy mennyiben jelent többet azoknál.
A relatív file szerkezeti felépítésében a rekord sorszáma tette lehetővé, hogy a rekord tartalmához hozzáférjünk. A rekord sorszáma tehát kulcs a rekord tartalmához.
Az ISAM file is kulcsos felépítésű, de a relatív file kulcsával szemben az ISAM file kulcsa a rekord alkotórésze, vagy pontosabban a rekord tartalmának logikai függvénye.
A relatív file-ban tárolt címek közül a név alapján csak akkor tudunk előkeresni egyet, ha rendelkezünk egy névjegyzékkel, azaz tudjuk, hogy az egyes nevek milyen kulcsú rekordban vannak elhelyezve. Az ISAM file-ban maga a név is lehet elérési kulcs, tehát nincs szükség külön névjegyzékre.
Elvileg a relatív file-t is felépíthetjük úgy, hogy a kulcsot a rekord tartalmából határozzuk meg.

Képezzünk kulcsot például a rekord első karakterének ASCII kódjából. Az ASCII kód 0 és 255 közé esik, tehát elvileg lehet rekordsorszám. Ez a megoldás mindaddig nem is jelent semmilyen problémát, amíg két egyforma karakterrel kezdődő rekordot nem akarunk tárolni (ami nagyon gyakran előfordulhat pl. számlanyilvántartásnál). A két azonos karakter azonos kulcsot eredményez, és a korábbi rekord tartalmát az új rekord tartalma kíméletlenül felülírja. Az ISAM file esetében a rekordok egymásra írása nem fordulhat elő, annak ellenére, hogy a kulcsmező két vagy annál több rekordban is lehet azonos. Ilyenkor a második, harmadik stb. azonos kulcsú rekord automatikusan egymás mögé, és nem egymásra íródik. Képzeljük el a címek ISAM tárolását úgy, hogy kódként a vezetéknév első tíz karakterét jelöljük meg. Meglepetésünkre soros visszaolvasásnál a rekordok a név első tíz karaktere szerint ABC sorrendben követik egymást. Azontúl, hogy az egyes rekordok bármely előre definiált kulcsmező szerint visszakereshetőek, a tárolás rendezett formában történik.
Az ISAM file-szervezést a kisebb személyi számítógépek lemezegységeinek operációs rendszere nem tudja megvalósítani.
Nem szeretnénk visszaélni az Olvasó türelmével, de meg kell említenünk, hogy van még néhány az ISAM-nál is magasabb szervezettségű file-típus.
Például a VSAM (Vertical Sequential Access Method). A VSAM file megengedi a két, sőt több kulcs egyidejű használatát. A címek nyilvántartásában pl. a vezetéknevet definiálhatnánk első, az irányítószámot második kulcsként. Az ilyen szerkezetű file-ban egyrészt mindkét név és irányítószám szerint is közvetlenül kereshetünk, másrészt ha úgy tetszik, lekérhetjük a teljes adathalmazt a vezetékneveken kívül az irányítószámok sorrendjében is. Tájékoztatásul megemlítjük, hogy a CBM 8032-es készüléken a SUPERKRAM nevű szoftver megvalósítja a VSAM file-kezelést.

A magasabb szintű adatfeldolgozás világában tett kirándulás után térjünk vissza a Commodore 64-es és a VC–1541-es realitásokhoz. A következő fejezetben bemutatjuk, hogy mi az a maximálisan elérhető „komfort", amit a VC–1541-es lemezegység a file-kezelésben adhat. Természetesen ötleteinkhez szubrutinokat is mellékelünk.

2.3.1.4 A QUISAM

A QUISAM file a szerzők saját találmánya. A szerzők a fejükbe vették, hogy a relatív- és soros tárolás előnyeit úgy egyesítik, hogy az eredmény egy olyan univerzális file-kezelési eljárás legyen, amelyben nem fordulnak elő – a VC–1541-es lemezműveleteit időnként jellemző – rendellenességek.
A QUISAM neve a QUasi-ISAM összetételből származik, és arra utal, hogy az új file szerkezete és adottságai némileg hasonlíthatóak az ISAM file szerkezetéhez és adottságaihoz. Ha az Olvasó nem sajnálja a fáradtságot és a szubrutinok begépelésén kívül az egyes fejezeteket is áttanulmányozza, nemcsak a QUISAM file felépítését, hanem a file-kezelés legmélyebb összefüggéseit is meg fogja érteni.
Foglaljuk össze a relatív file és soros file előnyeit, valamint hátrányait:

REL előny: Gyors hozzáférés az egyes rekordokhoz. Könnyen módosítható és bővíthető.
hátrány: A file rendszerint nagyobb területet foglal el, mint ami ténylegesen szükséges lenne a rekordok tárolásához. A rekordokat nem lehet utólag növelni. Az elérési kulcs csak a rekord sorszáma lehet.
SEQ előny: Folyamatosan keletkező adatok gyors tárolása. Dinamikusan bővíthető, azaz a file mérete az újabb adatok tárolása során automatikusan növekszik.

hátrány: A file-on belül nem lehet szabadon hozzáférni az adatokhoz. Nehezen módosítható.

Milyen tulajdonságokkal rendelkezzen egy általunk elképzelt file-típus? Lehetőleg tegyen eleget a következő elvárásoknak.

1. Szabad hozzáférés tetszőleges rekordhoz a rekordsorszám alapján.
2. Hozzáférési lehetőség az adatmező alapján.
3. A file dinamikus bővíthetősége.
4. A rekordok dinamikus bővíthetősége.
5. Elfogadható elérési idő.
6. Egyszerű használhatóság a BASIC nyelvből.

Hogy miként valósítottuk meg elképzeléseinket, azt továbbra is a címek nyilvántartásának példáján keresztül szemléltetjük. Először is egy úgynevezett törzsrekordot kell összeállítanunk. A törzsrekord tartalmazza esetünkben a nevet, címet stb. tehát mindazon adatokat, amelyek egy levél megcímzéséhez szükségesek. Hogyan képezzük a rekord kulcsát?
Az egyszerű emelkedő számozás nem jó, hiszen amellett egy névjegyzéket is tárolni kell. Az lenne célszerű, ha a kulcsot közvetlenül a vezetéknevekből képezhetnénk. A következő pontban bemutatjuk, hogyan lehet egy szöveges változóból egy lehetőleg egyedi kulcsot készíteni.

2.3.1.4.1 A HASH kód

Az előző alfejezetből világossá vált a feladat. Egy adathalmaz minden eleméhez egyértelműen hozzá akarunk rendelni egy egész számot, mégpedig úgy, hogy azt magából az adatból bizonyos műveletek elvégzésével állíthassuk elő. Ráadásul ahhoz, hogy minden adatot egyértelműen azonosítani tudjunk, nem engedhetjük meg, hogy két különböző adat azonos számot kapjon. Ezt az azonosító számot HASH kódnak nevezzük.
Nézzünk erre egy példát. Tegyük fel, hogy az ABC betűihez akarunk HASH kódot képezni. Erre a célra kiválóan alkalmas az ASC függvény, amely minden betűhöz egyértelműen egy kódot rendel. Ha a betűt az R$ változóban tároljuk, a kódot előállító BASIC utasítás: R=ASC(R$).
A hozzárendelés egyértelmű, minden kódból meghatározhatjuk visszafelé a kódhoz rendelt betűt.
Nagyon jó lenne, ha ilyen egyszerűen elő lehetne állítani egy adathalmaz HASH kódrendszerét. A leírt eljárás azonban sajnos még sántít egy kicsit. Bár képtelenség, a feladat elemzése kedvéért tegyük fel, hogy az ABC betűit egy file-ban akarjuk tárolni, és a kapott kód lesz annak a kódnak a sorszáma, amelyben a betűt tároljuk. Rögtön kiderül, hogy 1-től 64-ig a rekordok mind üresen maradnak, hiszen ezekhez nem tartozik betű. A file jó része kihasználatlanul marad.
Nem is beszélve arról, hogy ha kétbetűs szövegekhez próbálnánk egyértelmű kódokat rendelni a fenti módon, a kód már négyjegyű, és a tárolás még hézagosabb lenne.
Az egyértelműség mellett legalább ugyanilyen fontos követelmény az, hogy a kódok folyamatosan és ne hézagosan kövessék egymást. Próbálkozzunk meg egy másik képlettel:

```
R=VAL(RIGHT$(STR$(ASC(R$)),1))
```

A fenti képlet alapján R a betű ASCII kódjának első számjegye. Az R$="A" érték mellett R=5. Sajnos R$="K"-ra is ugyanezt kapjuk, tehát a kód nem egyértelmű.
Más a helyzet akkor, ha négy betűhöz rendelünk kódokat. Az eredmény egy 0 és 9999 közé eső szám. Az R$="FPZF" szöveghez a 0, az R$="EOYD" szöveghez a 9999 kód tartozik. A négyjegyű kódok képzésére a fenti képlet már nem alkalmas, a képzési szabályt a későbbiekben ismertetjük. Számunkra az a fontos, hogy hihetetlenül kicsi annak a valószínűsége, hogy két különböző szöveghez azonos kódokat kapjunk.
A valódi HASH kódokat az adatmező öt karakteréből szokták képezni. Mindig a tárolt adatok jellegétől függően kell eldönteni, hogy melyik öt karaktert válasszuk ki, melyik öt karakternél legkisebb az egyezés valószínűsége.
Eljutottunk addig, hogy van egy számítási eljárás, amely szöveghez egész számokat rendel kódként. De továbbra sem oldottuk meg a számtartomány kérdését. A fenti módon képzett négyjegyű kódokra gondolva a relatív file legnagyobb sorszámú rekordja a 9999. rekord lenne, ami merő képtelenség (ugyanakkor a rekordok óriási hányada üresen maradna).
Tegyük fel, hogy 100 címet fogunk tárolni.
A kódoknak tehát az 1-től 100-ig terjedő számtartományba kell esniük.
Induljunk ki először a vezetéknév első öt karakteréhez rendelt ötjegyű számból. Képezzünk belőle egy 0 és 1 közé eső számot, ebben a tartományban legtöbb ugyanis az értékes jegyek száma.
Próbálkozzunk meg azzal, hogy az ötjegyű számot elosztjuk 100-zal, vesszük a kapott szám tizedes részét, és ezt ismét felszorozzuk 100-zal. A számítás a 75460-ra a 754,6-et, majd a 60-at adná. A 60 már megfelelő 1 és 100 közé eső rekordsorszám lenne, de sajnos a módszer mégsem jó. Minden 60-ra végződő ötjegyű számból ugyanis azonos rekordszámot kapnánk. Ha azonban a 75460-at 101-gyel osztjuk el, az eredmény sokkal ígéretesebbnek bizonyul: 747,1281713. Vágjuk le a hányados tizedes részét (0,1281713), szorozzuk meg 100-zal (12,81713), vegyük a kapott szám egész részét. Az eredmény 12.
Rendkívül kicsi a valószínűsége annak, hogy ez a számítás két különböző vezetéknévhez azonos 1 és 100 közé eső számot rendel. Miért pont 101-gyel osztottunk? Ez nem véletlen. A 101 ugyanis prímszám, amely csak 1-gyel és önmagával osztható maradék nélkül. A prímszámokkal való osztás hányadosa emiatt a számjegyek rendkívül változatos sorozata.
Általános szabály, hogy osztóként a maximális rekordszámnál nagyobb első prímszámot használjuk, tehát például egy 1000 rekordból álló file esetén az 1009-et.
A következőkben bemutatunk egy kis szubrutint, amely előállítja a HASH kódokat. A kód képzéséhez meg kell adni a teljes rekordot, a rekordon belül a kódkritérium kezdetét, a kód hosszát, a prímszámot és a file méretét rekordokban.
A szubrutin listája:

```
P10

58800 :
58801 REM **** A KODOK KIALAKITASA ****
58802 :
58820 HC$=MID$(QD$,QK%,QH%)+"      "
58840 HC=0:FORL1=0TO4
58860 HC=HC+VAL(RIGHT$(STR$(ASC(MID$(HC$,L1+1,1))),1))*10^L1
58880 NEXTL1
58900 HC=INT((HC/QP-INT(HC/QP))*QB)+2
58920 RETURN

READY.
```

A változók jelentése:

QD$ A teljes rekord
QK% A kódkritérium rekordon belüli pozíciója
QH% A kód hosszúsága. Ebben a rutinban mindig öt karakterrel képezzük a HASH kódot, függetlenül a QH%-ban lévő értéktől. A tényleges hosszúságnak a QUISAM file-nál lesz jelentősége. A QUISAM példában a kód a vezetéknév teljes hosszával azonos, vagyis QK% = 16-nál 12.
QB A file mérete (maximum hány rekordot tartalmaz)
QP A kód kiszámításához szükséges prímszám
HC$ A kódot meghatározó karaktersorozat
HC A kész rekordszám (HASH kód)

A program leírása:

58820 A HC$-t a QD$-ból a QK% és QH% paraméterekkel leválasztjuk és kiegészítjük öt karakterre, ha QH% kisebb, mint öt.

58840-58880 Az ötjegyű számot előállító ciklus. A képlet csak látszólag bonyolult: Balról kezdve képezzük minden karakter ASCII kódját, levágjuk a jobboldali számjegyét, megszorozzuk 10 L1-edik hatványával, végül a kapott értéket a HC változóban gyűjtjük. Az eredmény egy maximum ötjegyű szám, melyben a számjegyek sorrendje fordított ahhoz képest, mint amit a karaktersorrend alapján várnánk.
Ennek egyszerűen az a magyarázata, hogy a kódmezőt ugyan balról bontjuk karakterekre, de szorzótényezőként 10 legkisebb hatványával kezdünk. A megoldás így egyszerűbb, a HASH kód szempontjából pedig a számjegyek sorrendje lényegtelen.

58900 A számítási algoritmus nem zárja ki, hogy az eredmény 0 legyen, a 0 azonban nem lehet rekordszám, ezért HC-t azonnal megnöveljük kettővel. Így az első rekord kulcsa 2 lesz. Azért nem írunk az 1-es kulcsú rekordra, mert erre szükségünk lesz a QUISAM file kezelésénél. Az első rekordra írjuk például az olyan leíró paramétereket, mint QB, QP, QK%, QH%. Ezekre a paraméterekre ugyanis a későbbi feldolgozás során mindig szükségünk lesz, és ha nem tárolnánk a program minden futásakor, a program használójának kellene ezeket emlékezetből beírni.

Kapcsoljuk össze ezt a szubrutint a korábban bemutatott relatív file-kezelő szubrutinokkal, és írjunk belőle egy új címnyilvántartó programot. Ha a rekordok visszakereséséhez ugyanazt az algoritmust használjuk, mint amit a kódok kiszámításánál használtunk, akkor a vezetéknév alapján bármely rekordot visszakereshetünk. Ha a kereséshez csak a vezetéknevet adjuk át a rutinnak, akkor a QK% változóba 1-et kell írni.
Kérjük az Olvasót, hogy próbálgassa egy kicsit az elkészült programot. Biztosak vagyunk abban, hogy nem unalmas foglalatosság. A használt algoritmus alapján előfordulhat, hogy különböző nevekre ugyanazt a rekordszámot adja a program. A következő fejezetben kiküszöböljük ezt a hibát. Addig is egy ötlet: minél nagyobbra méretezzük a file-t, annál kisebb a valószínűsége az azonos kódoknak.

2.3.1.4.2 A file mérete

Az adatkezelésnél nagyon fontos tényező a file mérete. Mi lehet az oka annak, hogy az előző fejezetben alkalmazott számítások során azonos kódok is előfordultak?
Az egyik ok feltétlenül az, hogy egy ötjegyű számot kevesebb jegyű számmá alakítunk át annak érdekében, hogy megfelelő rekordszámokat kapjunk. Menet közben értékes számjegyeket vágunk le, így előfordulhat, hogy az eredetileg különböző szövegekből azonos kód képződik. Ha megnöveljük a file méretét, a rekordszámok tartománya nagyobb lesz, így az átalakításnál kevesebb információ vész el. Az Olvasó biztosan érzi, hogy a file méretének növelése nem jelenthet igazán megoldást, hiszen így egyre több hely marad a lemezen kihasználatlanul.
A kódok azonosságának másik oka az lehet, hogy az egyes karakterek decimális kódjából csak az utolsó számjegyet használjuk. Mi tehát a teendő?
Az azonos kódok elkerüléséhez figyelhetjük a programból, hogy az éppen kiszámított kódú rekord foglalt-e már, és ha igen, vesszük a következő szabad kódot.
Az ötlet nem rossz, éppen csak a dolog lényegét kell feláldoznunk érte, hiszen a gyors visszakeresés lehetőségét az algoritmusba való durva beavatkozással el is ronthatjuk. Hogyan kellene ez esetben visszakeresni egy kódmezőt?
Először el kell olvasni az egész rekordot, a kódmezőt össze kell hasonlítani a keresett kódmezővel, és egyenlőtlenség esetén tovább kellene lépni a következő rekordra egészen addig, amíg vagy megtaláljuk a keresett kódmezőt, vagy egy szabad rekordot találunk, ami azt jelenti, hogy a keresett kódmező nem létezik.
Ez a keresési mód még mindig gyorsabb, mint a soros keresés. A file rekordjainak sűrűségétől függően maximum hat hozzáféréssel kell számolni addig, amíg a keresett rekordot megtaláljuk, ami egészen gyorsan megy, ha a lemezegység lendületbe jött.
Ha bele is törődünk a kissé körülményes keresési eljárásba, még mindig nem oldottuk meg az üresen hagyott rekordok problémáját. A következő elképzelés meghozza a végleges megoldást: ha már számoltunk azzal, hogy a keresést több lépésben végezzük, tervezzük eleve kisebbre a file-t, mégpedig olyan kicsire, hogy az lehetőleg tele legyen, vagy minimális üres rekordot tartalmazzon.
Igen ám, gondolja az Olvasó, de mi legyen az azonos HASH kódú rekordokkal?
A válasz egészen egyszerű: ezek számára készítünk egy második file-t, amelybe a kimaradt rekordokat szép sorjában beleírjuk. Az elképzelésnek sok előnye van. Tökéletesen mindegy, hogy az első melléfogás után ugyanabban a file-ban kell-e a rekordot tovább keresni, vagy egy másik file-ban, tehát az eredeti helyzeten nem rontottunk.
Ugyanakkor a másodlagos file-t nem kell előre feleslegesen felduzzasztani, ráérünk akkor bővíteni, amikor a bővítés igénye felmerül, vagyis amikor egy kód megismétlődik.
A végső megoldás tehát: egy sűrűn kitöltött elsődleges file, amiben azonnal megtaláljuk, amit keresünk, és egy dinamikusan növekvő másodlagos file, amelyben a néhány kivételes rekord található. Természetesen abban az esetben, ha előzetesen rosszul becsültük meg az elsődleges file méretét, kicsire terveztük, és emiatt a másodlagos file gyorsan duzzad, a keresés pedig egyre lassúbb, feltétlenül új elsődleges file-t kell készíteni.
Az nem megy, hogy egyszerűen megnöveljük a meglévő file-t és az új rekordokat új képlettel kódoljuk. Nyilvánvaló, hogy ekkor a régi rekordokat a program sohasem találná meg.

A gyanakvóbb Olvasóban még biztosan maradt némi kétség az ismertetett módszerrel szemben.
Sajnos jogosan. Mi a garancia ugyanis arra, hogy a legelőrelátóbb tervezés ellenére az adathalmaz növekedtével nem duzzad fel a másodlagos file? Semmi. Semmilyen egzakt

bizonyítást nem adtunk arra nézve, hogy azonos kódok nem, vagy csak nagyon ritkán fordulnak elő. Akkor viszont számolnunk kell azzal, hogy a másodlagos file-ban egyszercsak már több mint 100 rekord lesz, és ha a rekordok közül éppen a 100.-at keressük, menthetetlenül végig kell olvasnunk az előtte álló 99-et. Ugyanott tartunk, mintha soros file-lal dolgoznánk.
Holott, ha az Olvasó kicsit is figyelt arra, amit olvasott, még talán emlékszik ama ígéretünkre, hogy minden rekordot maximum hat lépésben meg fogunk találni. Nem füllentettünk, valóban így is lesz, ha még van egy kis türelme az Olvasónak kivárni elméletünk végső kifejletét.
A megoldás egészen egyszerű: az elsődleges file minden rekordjában tartsunk fenn egy helyet azon rekord sorszámának a tárolásához, ahová az első kódismétlés során a primer file-ból kieső rekord kerül. Ez a rekordsorszám nyilván a másodlagos file-ra vonatkozik. Hogyan történik most a keresés? Összehasonlítjuk a keresett adatmezőt a karakterekből kiszámolt sorszámú rekord megfelelő adatmezőjével. Ha a két mező nem azonos, akkor kiolvassuk ugyanebből a rekordból a folytató rekordsorszámot. A másodlagos file-t tehát nem kell elejétől végigolvasni, csak ettől a rekordszámtól kezdve.
És hogy a pohár beteljék: a másodlagos file adott rekordjában is elhelyezünk egy mutatót arra az esetre, ha az azonos kód nem egyszer fordult volna elő. Ez a mutató megadja, hogy a másodlagos file-ban hol található a következő ugyanilyen kódú rekord és így tovább!
Az azonos kódú rekordokat egyszerűen összeláncoltuk. Egy rekord keresésekor nem kell egyik file-t sem sorosan olvasni, csak a láncolás mentén folyamatosan haladni.
Azt, hogy egy rekord azonosító kódja még nem fordult elő egynél többször, jelezzük egy, az elsődleges file-beli rekordba írt 0 rekordsorszámmal. A keresés során a 0 észlelése azt jelenti, hogy a keresett adatmező nem létezik.
Még mindig ne adja fel az az Olvasó, aki nem tudta követni a gondolatmenetet, hanem nézze meg a következő táblázatot:

Elsődleges file			**Másodlagos file**		
A HASH kódból nyert rekordszám	A folytató rekordszám	Adatok	Rekordszám	A folytató rekordszám	Adatok
:		:	:	:	:
:		:	:	:	:
2	4	Szabó 1	2	3	Tóth 2
3			3	9	Tóth 3
4	5	Kovács 1	4	6	Szabó 2
5			5	0	Kovács 2
6			6	0	Szabó 3
7	7	Vas 1	7	8	Vas 2
8			8	0	Vas 3
9					
10	2	Tóth 1			

Az ábrán jól látható, hogy az elsődleges file-ban még vannak hézagok, a másodlagos file azonban tele van, mivel a másodlagos file csak szükség esetén bővül.
A két file-ban az első rekord üres. Erre a későbbiekben lesz szükségünk, hogy egy kevés adminisztrációt végezhessünk.
Most feltesszük az i-re a pontot:
Időnként előfordulhat, hogy egy bevitelhez néhány megjegyzést kell fűzni. Ha pl. a címnyilvántartás vevők címeit tartalmazná, jó lenne a rekordban még további információkat is elhelyezni,

például azt, hogy a vevő megbízhatatlan, vagy mindig pontosan fizető személy. Természetesen helytelen volna az ilyen járulékos információk tárolására a rekordméretet megnövelni, mert az ilyen megjegyzések nem minden vevőnél szerepelnek és a megjegyzések hosszúságát sem lehet előre pontosan megbecsülni. Ha erre külön helyet terveznénk, a helypocsékolás óriási méreteket öltene.

Az eddigi tapasztalataink szerint mi sem kézenfekvőbb, mint egy kis rekordhosszúságú harmadik file-t készíteni. A file szerkezete megegyezik a másodlagos file szerkezetével, tehát bővítésére csak szükség esetén kerül sor. A rekord hosszát az átlagos hosszúságú megjegyzések hossza alapján kell meghatározni.

A hozzáférés is ugyanúgy történik, mint a másodlagos file esetében. Ehhez azonban a törzsrekordban egy további azonosítót kell nyilvántartani, amely 0, ha nincs, és a megjegyzés sorszáma a harmadik file-ban, ha van megjegyzés.

A törzsrekordhoz tartozó megjegyzéseket egymáshoz láncolhatjuk ugyanúgy, mint a másodlagos file rekordjait. Ha egy adatrekord hosszabb lenne, mint az eredeti rekordhossz, a fenti módon egyszerűen hozzáláncolhatunk egy második rekordot.

A láncolás további előnye az időbeli rugalmasság. Egy rekordhoz még évek múltán is fűzhetünk megjegyzést és bármikor újra visszakereshetjük.

Az új, és immár végleges file-szerkezetet mutatja a következő táblázat. Már csak egy kis lépés van hátra az önmagát kezelő QUISAM file-ig. Az első program, amit szeretnénk bemutatni, egy QUISAM file felépítése. Ez az Olvasótól már maximális aktivitást kíván, mert ettől kezdve minden (majdnem minden) automatikusan történik.

Elsődleges file

Rekordszám	A másodlagos file rekordszáma	Megjegyzések rekordszáma	Sorszám	Adatok
2	2	3	0	Tóth 1
4	0	3	0	Vas 1
15	4	8	0	Kis 1

Másodlagos file

1	6			
2	3	5	1	Kovács 1
3	0	6	2	Kovács 2
4	5	9	1	Kis 2
5	0	10	2	Kis 3

Megjegyzések file

		Adatok
1	12	
2	0	telefon 692-704
3	4	csinos lány
4	0	ebédre meghívni
5	0	100 Ft-tal tartozik
6	7	XII. 22-én meglátogatni
7	0	virágot vinni
8	0	hamupipőke
9	0	pincér a „Bikában"
10	11	I. 20-án születésnap
11	0	nem elfelejteni!

A program listája:

```
P11

59000 :
59001 REM **** LETREHOZNI ****
59002 :
59020 PRINTCHR$(147)"A QUISAM FILE LETREHOZASA"
59030 PRINT
59040 INPUT"A FILE NEVE (MAX. 8 KAR.)";QN$
59050 PRINT:IFLEN(QN$)>8THEN59040
59060 INPUT"A REKORDOK BECSULT SZAMA:";QB
59070 PRINT
59080 INPUT"A REKORDOK HOSSZA (MIN 4, MAX. 249)";GL%
59090 PRINT:IFQL%<40ORQL%>249THEN59080
59100 INPUT"AZ ALREKORD HOSSZA (MIN. 2, MAX. 252)";QX%
59110 PRINT:IFQX%<20ORQX%>252THEN59100
59120 INPUT"A KULCS KEZDO POZICIOJA";QK%
59130 PRINT
59140 INPUT"A KULCS HOSSZA";QH%
59150 IF(QK%+QH%-1)>GL%THEN59120
59160 QB=INT(QB/6)+1:QP=QB+1
59180 IGQPAND1THEN59200
59190 QP=QP+1:QX=2:QS=2
59200 OPEN15,8,15
59210 L1=3
59220 IFQP/L1=INT(QP/L1)THENQP=QP+2:GOTO59210
59240 L2=INT(QP/L1):L1=L1+1:IFL1<L2THEN59220
59260 DN$=QN$+".PR":R=QB+2:RL%=QL%+6:GOSUB63800
59280 R$=CHR$(QL%)+CHR$(QX%)+CHR$(QK%)+CHR$(QH%)
59300 RH%=QB/256:RL%=QB-256*RH%
59320 R$=R$+CHR$(RH%)+CHR$(RL%)
59340 RH%=QP/256:RL%=QP-256*RH%
59360 R$=R$+CHR$(RH%)+CHR$(RL%)
59380 R=1:GOSUB63000
59400 R$=CHR$(0)+CHR$(0)+CHR$(0):R$=R$+R$
59460 PRINT#15,"P"CHR$(6)CHR$(2)CHR$(0)CHR$(1)
59480 FORL1=2TOQB+1
59500 PRINT#6,R$:NEXTL1:CLOSE6
59520 DN$=QN$+".SE":R=1:RL%=QL%+6:GOSUB63800
59540 R$=CHR$(0)+CHR$(2):GOSUB63000
59560 DN$=QN$+".EX":RL%=QX%+3:GOSUB63800
59580 GOSUB63000:RETURN

READY.
```

A programban használt változók jelentése:

QN$	A QUISAM file neve. A név nyolc karakterből állhat, amit a program automatikusan kiegészít két karakterrel: ".pr" az elsődleges file, ".se" a másodlagos file és ".Ex" a megjegyzés file jelölésére.
QB	A szükséges rekordok becsült száma.
QL%	Egy rekord hosszúsága. Az általunk javasolt szerkezetű címfile esetén ez 67.
QX%	Egy megjegyzés hossza.
QK%	A GK% változó már ismert a HASH kód rutinból. Értéke megadja, hogy a rekord melyik pozíciójában kezdődik a kódfogalom. A címrekordban ez 16.
QH%	A kód hossza, nálunk 12.
QP	A maximális rekordszámhoz tartozó prímtényező, amit a rutin számol ki.

A program leírása:

59020–59150 A leíró paraméterek beolvasása klaviatúráról, párbeszéd formájában.

59160 Feltételezzük, hogy az Olvasó megelégszik egy maximum hatszoros keresőhozzáféréssel. Ezért az elsődleges file méretéhez megadott rekordok számát elosztjuk hattal. Ha idővel a hozzáférések túl sok ideig tartanának, egyszerűen állítsuk be ízlés szerint az osztó értékét. Az így kapott számot eggyel növeljük, mivel a file kezeléséhez még egy rekordra szükségünk van.

59180–59240 A QP prímszám kiszámítása. Egyszerűen kipróbáljuk, hogy a szám (QB + 1-gyel kezdünk) osztható-e maradék nélkül egy másik számmal (2-vel kezdünk). Ha igen, akkor ez nem prímszám és megpróbálkozunk a következő egész számmal. Az osztás kísérletét addig folytatjuk, amíg elérünk egy felső határt, amelynél nagyobb osztója a számnak matematikailag nem létezik.

59260 Az elsődleges file előkészítése.

59280–59380 Az elsődleges file első rekordjába felírjuk a file paramétereit, hogy ezeket a változókat ne kelljen manuálisan őrizni.

59400–59500 Az elsődleges file minden rekordjába hat bináris nullát írunk. Erre később az üres rekord felismeréséhez lesz szükség.

59520–59580 A másodlagos- és a megjegyzés file-ba felírjuk az első rekordot. Ebbe a rekordba írjuk mindig az első szabad rekord számát, ami esetünkben 2.

Legcélszerűbb, ha a már meglévő QUISAM file megnyitására szolgáló programmal folytatjuk, hogy az Olvasó lássa, használjuk a file első rekordjaiban elhelyezett paramétereket.

```
P12

58000 :
58001 REM **** A QUISAM FILE MEGNYITASA ****
58002 :
58010 CLOSE6:CLOSE15:OPEN15,8,15:RZ$=""
58020 DN$=QN$+".PR":R=1:RC%=8:GOSUB63500
58040 QL%=ASC(LEFT$(R$,1))
58060 QX%=ASC(MID$(R$,2,1))
58080 QK%=ASC(MID$(R$,3,1))
58100 QH%=ASC(MID$(R$,4,1))
58120 QB=ASC(MID$(R$,5,1))*256+ASC(MID$(R$,6,1))
58140 QP=ASC(MID$(R$,7,1))*256+ASC(RIGHT$(R$,1))
58160 DN$=QN$+".SE":RC%=2:GOSUB63500
58180 QS=ASC(LEFT$(R$,1))*256+ASC(RIGHT$(R$,1))
58200 DN$=QN$+".EX":GOSUB63500
58220 QX=ASC(LEFT$(R$,1))*256+ASC(RIGHT$(R$,1))
58240 RETURN
58300 :
58301 REM **** A QUISAM FILE LEZARASA ****
58302 :
58320 CLOSE6:RETURN

READY.
```

A program változói:

QN$ Az egyetlen változó, amelynek értéket kell adni a rutin hívása előtt. Tartalma a file neve.

QS Új változó. A másodlagos file első szabad rekordjának sorszáma. Ez a szám a file első rekordjából származik.

QX Jelentése ugyanaz, mint a QS változónak, de a megjegyzés file-ra vonatkoztatva.

A program leírása:

58010–58140 Beolvassuk az első rekord első nyolc byte-ját. Tartalmukat tároljuk azokban a változókban, amelyeket a file paraméterei számára tartunk fenn. Így biztosítjuk, hogy a QUISAM file mindig azokkal a paraméterekkel dolgozzon, amelyeket a file felépítése pillanatában rögzítettünk.

58160–58180 Beolvassuk a másodlagos file első szabad rekordjának sorszámát és a QS-ben tároljuk.

58200–58220 Mint fent, de a megjegyzés file-ra és a QX változóra vonatkoztatva.

58300–58320 Az adatcsatornát itt zárjuk le, hogy az Olvasónak csak itt kelljen változtatnia, ha a REL-hozzáféréseknél a logikai file- számot módosította.

Most már tulajdonképpen minden készen áll ahhoz, hogy az Olvasó kipróbálhassa a QUISAM file kezelését.
A feladat mindössze annyi, hogy az eddig bemutatott rutinokat – a képernyőmaszktól kezdve egészen idáig – az Olvasó egymással ésszerűen összekapcsolja és ellássa a szükséges változókkal.

Valami fontos dolog mégis kimaradt. Gondoljuk csak át nyugodtan!
????????????
Mit csinálunk akkor, ha a másodlagos file-ból egy rekord fölöslegessé vált és törölni akarjuk? Mi történik ilyen esetben a láncolt mezőkkel?
Nem lenne szép tőlünk, ha magára hagynánk az Olvasót ezzel a problémával. Ki gondolta volna, hogy miután minden olyan pompásan megy, éppen a rekord törlése okoz majd gondot?
Kiderült, hogy a QUISAM file témaköre még egyáltalán nincs lezárva!

2.3.1.4.3 Egy rekord törlése a QUISAM file-ból

A soros file-ból csak nagyon körülményesen tudunk egy rekordot törölni. Az egész file-t át kell másolnunk egy új file-ba, a törlendő rekordok kihagyásával. Folyamatosan olvasva a régi file rekordjait, meg kell vizsgálni, hogy az éppen beolvasott rekord azonos-e a törlendő rekorddal.
Ha igen, akkor ezt a rekordot nem másoljuk át az új file-ba. Ha nem, akkor átmásoljuk. Egy hosszabb, pl. 100 rekordból álló file esetén egyetlen rekord törlése rengeteg időt vesz igénybe.
Valamivel egyszerűbb a törlés, ha relatív file-lal dolgozunk. A törlendő rekordot nem távolítjuk el fizikailag a file-ból, hanem közvetlen eléréssel „nem létező" rekordnak nyilvánítjuk. Megegyezés szerint egy olyan karaktert, vagy szöveget írunk bele, amelyből minden, a file-t használó program tudni fogja, hogy a rekordban nincs érvényes adat. Beleírhatjuk például azt, hogy TÖRÖLVE. A feladatot látszólag megnyugtatóan meg is oldottuk.
Miért csak látszólag?
A törölt rekordok ugyanis fizikailag bent maradnak a file-ban, és ha nem használjuk ezeket a rekordokat új adatok tárolására, akkor a file egy idő után tele lesz érvénytelen adatokkal.

A bővítés során meg kellene vizsgálni, hogy van-e már érvénytelen rekord, és ha van, az új adatokat ebbe a rekordba kellene beleírni ahelyett, hogy ténylegesen új rekordot fűznénk a file-hoz. Gondoljuk azonban végig, hogy mivel jár ez a vizsgálat? Rengeteg fölösleges munkával. Akár volt törlés időközben, akár nem, végig kell olvasnunk esetenként az egész adathalmazt, ha biztosak akarunk lenni abban, hogy jogosan bővítjük a file-t, azaz valóban nincs szabad rekord.
Hogyan oldottuk meg a QUISAM file rekordjainak törlését? A file felépítésekor gondoskodtunk arról, hogy a másodlagos- és a megjegyzés file első szabad rekordjának sorszámát mindig ismerje a rendszer. Az elsődleges file első rekordjában tárolt paramétereket a program elindításakor rögtön beolvassuk a QS és a QX változókba.
Másrészt az elsődleges file minden rekordjának első adatmezejébe már a file létrehozásakor beírtunk egy CHR$(0) karaktert, ezzel jelezve, hogy a rekord még szabad. Ha az elsődleges file-ba új rekordot akarunk beírni, kiszámítjuk a HASH kódot, beolvassuk a kapott sorszámú rekordot, és attól függően, hogy az első adatmezőben CHR$(0) van vagy sem, az új adatokat beleírjuk a rekordba, vagy tovább megyünk szabad helyet keresni a másodlagos file-ban. Következésképpen az elsődleges file-ban egy rekordot rendkívül egyszerűen törölhetünk, nem kell mást tenni, mint a rekord első adatmezőjét CHRS(0)-val felülírni.
Ha a másodlagos- vagy a megjegyzés file-ban kell a rekordot törölni, akkor a kérdéses rekordhoz hozzá sem kell nyúlni. A rekord sorszámát a QS, ill. a QX változóba beírva ugyanis az adott rekord automatikusan felszabadul.
Ha egymás után több rekordot akarunk törölni, akkor a felszabaduló rekordokat a foglalt rekordokhoz hasonlóan egymáshoz láncolhatjuk. Mindig a legutoljára felszabadult rekord sorszámát tároljuk a QS változóban.
Az elsőként törölt rekord sorszáma első lépésben a QS változóba kerül. A következő törlésnél a most felszabaduló rekord sorszáma kerül a QS-be, a QS korábbi tartalmát pedig beleírjuk az éppen felszabaduló rekord első adatmezőjébe, és így tovább.
Az elmondottakat szemlélteti a következő két táblázat. A címnyilvántartásból töröljük a "Kis 2" kódmezővel jelzett rekordot. A törlendő rekord a másodlagos file-ban van. Törlés előtt a másodlagos file első szabad rekordja a 6. rekord volt, ezt a számot találtuk a másodlagos file 1. rekordjában. Most az 1. rekordba 4-et írunk, mert a felszabadított rekord a másodlagos file 4. rekordja. A szabad helyet jelző korábbi érték (6) pedig bekerül a 4. rekordba folytató sorszámként.
A "Kis 2" kódmezővel ellátott rekordhoz tartozó megjegyzést hasonlóan kell törölni.
A megjegyzés file korábbi első szabad rekordjának sorszáma (12) bekerül a 9. rekord első adatmezejébe folytató sorszámként, míg az 1. rekordban az első szabad rekord sorszáma 9 lesz (QX). Természetesen közben megváltoztatjuk az elsődleges file "Kis 1" kódmezővel ellátott rekordjában is a folytató sorszám értékét, amely most a "Kis 3" kódmezővel ellátott rekordra mutat, értéke tehát 4 helyett 5 lesz.

Elsődleges file

Rekordszám	A másodlagos file rekordszáma	Megjegyzések rekordszáma	Sorszám	Adatok
2	2	2	0	Tóth 1
4	0	3	0	Vas 1
15	5	8	0	Kis 1

Másodlagos file

Rekordszám	A másodlagos file rekordszáma	Megjegyzések rekordszáma	Sorszám	Adatok
1	4			
2	3	5	1	Kovács 1
3	0	6	2	Kovács 2
4	5	6	1	0
5	0	10	2	Kis 3

Megjegyzések file

		Adatok
1	9	
2	0	telefon 692-704
3	4	csinos lány
4	0	ebédre meghívni
5	0	100 Ft-tal tartozik
6	7	XII. 22-én meglátogatni
7	0	virágot vinni
8	0	hamupipőke
9	12	
10	11	születésnap I. 20.
11	0	nem elfelejteni!

Elsődleges file

2	2	2	0	Tóth 1
4	0	3	0	Vas 1
14	4	9	0	Nagy 1
15	5	8	0	Kis 1

Másodlagos file

1	6			
2	3	5	1	Kovács 1
3	0	6	2	Kovács 2
4	0	12	1	Nagy 2
5	0	10	2	Kis 3

Megjegyzések file

		Adatok
1	13	
2	0	telefon 692-704
3	4	csinos lány
4	0	ebédre meghívni
5	0	100 Ft-tal tartozik
6	7	XII. 22-én meglátogatni
7	0	virágot vinni
8	0	hamupipőke

Rekordszám	A másodlagos file rekordszáma	Megjegyzések rekordszáma	Sorszám	Adatok
9	0	jó pástétomok		
10	11	I. 20-án születésnap		
11	0	nem elfelejteni!		
12	0	véreshurka nem jó!		

Eddig még nem esett szó a táblázatok utolsó oszlopának jelentéséről. Az utolsó oszlop az elsődleges file minden rekordjában 0, a másodlagos file rekordjaiban pedig az azonos HASH kóddal nyilvántartásba vett adatok növekvő sorszáma.
Az adatok és a rekordok sorszámainak hozzárendelése nem egyértelmű, hiszen mint tudjuk, különböző adatokhoz azonos kódot is kaphattunk (pontosan ezért volt szükség a másodlagos file-ra).
Azonban a HASH kódokhoz hozzáfűzzük az utolsó oszlopban álló számokat (egyedi HASH kód esetén 0-t), akkor már minden adathoz különböző számérték tartozik.
A következő táblázat alapján, ha az utolsó oszlopban maximum kétjegyű számokat engedünk meg, a Kovács 1 kódmezőhöz az 1500, a Kovács 2 kódmezőhöz az 1501-es azonosítószám tartozik.
Az adatrekordok egyértelmű azonosítószámait sok mindenre fel lehet használni. Ha például a QUISAM file-ban különböző anyagokat tartunk nyilván, az anyagnyilvántartásban kereshetünk az egyedi raktári szám alapján is. A QUISAM rendszer megkíméli a használót attól, hogy mindig új egyedi raktári számokat kelljen – a tévedés lehetőségét sem kizárva – gyártania. A QUISAM biztosan nem téved. A törlést követő adatbevitelt a táblázattal szemléltetjük. A Nagy 1 az elsődleges file első szabad rekordjába kerül, mert egyedi HASH kódot kapott. A Nagy 2 kódmezős adatok azonban a törölt Kis 2 kódmezős rekord helyére kerülnek a másodlagos file-ba. Az új bevitel után a másodlagos file első szabad adatterülete ismét a 6. rekord lesz.
Az új megjegyzést hasonlóan tároljuk. A Nagy 1 kódmezőhöz tartozó megjegyzés a törölt Kis 2 kódmezővel ellátott rekord megjegyzése helyére kerül, míg a Nagy 2-höz fűzött megjegyzés a file-t bővíti, hiszen nincs több törölt rekord a file-ban.
A törlő szubrutin tanulmányozása előtt nézze át az Olvasó a P 13-as szubrutint, amelyben megmutatjuk, hogyan lehet az adatrekordból közvetlenül, a rekord teljes beolvasása nélkül a kódmezőt beolvasni. A kódmező közvetlen elérésének a file-kezelés minden fázisában nagy jelentősége van. A szubrutin felhasználja az RC% paramétert. Az RC% változóval már a relatív file felépítése során találkozott az Olvasó, de akkor még nem tudtuk megindokolni, hogy miért van rá szükség.

A szubrutinban használt változók:

QT$ A file nevének kiegészítése (pl. .PR)
QT A rekord sorszáma
QY$, QZ$, QY, QZ,
Az alárendelt file-ok neve, és az alárendelt file-okon belüli rekordsorszámok, amelyekre a láncolás során szükségünk van.
Q1, Q2, Q3, Q4
A rekord első adatmezői. Q4 az első hasznos karakter ASCII kódja. Ilyen tárolás mellett könnyű lesz pl. a Q4=0 vizsgálata.

Q1$, Q2$, Q3$
: A Q1, Q2, Q3 változóknak megfelelő karakterek. Karakteres tárolásuk megkönnyíti a rekord visszaírását.

Q5
: A szabad tárkapacitás figyelésére szolgáló segédváltozó. Jelentése a programleírásból kiderül.

```
P13

58500 :
58501 REM **** ELOKESZITES ****
58502 :.
58510 QZ$=QY$:QY$=QT$:QZ=QY:QY=QT
58520 DN$=QN$+QT$:R=QT:RC%=6:GOSUB63500
58530 IFVAL(F1$)<>0THENRETURN
58540 Q1$=LEFT$(R$,2)
58560 Q1=ASC(LEFT$(Q1$,1)*256+ASC(RIGHT$(Q1$,1))
58580 IFQT$=".EX"THENRETURN
58600 Q2$=MID$(R$,3,2)
58620 Q2=ASC(LEFT$(Q2$+CHR$(0),1))*256+ASC(RIGHT$(Q2$,1)+CHR$(0))
58640 Q3$=MID$(R$,5,1):Q3=ASC(Q3$+CHR$(0))
58660 Q4=ASC(RIGHT$(R$,1)+CHR$(0))
58680 RETURN

READY.
```

A program leírása:

58510
: A paraméterek átvétele. QZ$ ill. QZ mindig az előző hívásban szereplő paramétereket tartalmazza.

58520
: A relatív file olvasásához szükséges paraméterek felépítése. A kódmező mindig a rekord első 6 karaktere (RC% = 6).

58530
: Ha a rekordot nem lehet beolvasni, mert az olvasás során hiba lépett fel, visszatérünk a főprogramba (a hiba oka pl. az, hogy a rekord nincs előkészítve).

58540–58660
: A beolvasott karaktereket és kódokat szétszedve a G1–G4 ill. a Q1$–Q3$ változókban tároljuk.

A következő P 14-es szubrutin elvégzi a rekord törlését.
A szubrutin megértéséhez még néhány megjegyzés:
Az adatbázist úgy építettük fel, hogy a HASH kódok között egyformák is előfordulhatnak. Előzetes vizsgálat, egyértelmű azonosítás nélkül nem szabad rekordot törölni a file-ból. Ezért törlés előtt előállítjuk az előző fejezetben említett egyértelmű azonosító számot.
A kódmező alapján kiszámítjuk a rekordsorszámot, a sorszám alapján beolvassuk a rekord tartalmát, és abból előállítjuk az egyértelmű azonosítót.
Az alábbi szubrutinban találkozunk egy eddig ismeretlen változóval, amelynek a szubrutin hívása előtt értéket kell adni (QN).

```
P14

55000 :
55001 REM **** A BEVITEL TORLESE ****
55002 :
55020 QN%=VAL(RIGHT$(STR$(QN),2))
55040 QT=(QN-QN%)/100
55060 QT$=".PR":QN=1:GOSUB58500
```

```
55080 IFQN%<>0THENQT=Q1:QN=0:GOTO55300
55100 IFQ4=0THENRETURN
55120 R$=Q1$+CHR$(0)+CHR$(0)+Q3$+CHR$(0)
55140 GOSUB63000:IFQ1=0ANDQT$=".PR"THEN55170
55150 DN$=QN$+QZ$:R=QZ:RC%=255:GOSUB63500
55160 R$=Q1$+MID$(R$,3,QL%+3):GOSUB63000
55170 ONQNGOTO55400
55180 RH%=QS/256:RL%=QS-RH%*256:DN$=QN$+QY$:R=QY
55190 R$=Q1$+CHR$(RH%)+CHR$(RL%)+Q3$+CHR$(0)
55200 GOSUB63000:QS=QT
55220 RH%=QS/256:RL%=QS-RH%*256
55240 R=1:R$=CHR$(RH%)+CHR$(RL%)
55260 GOSUB63000:GOSUB55400
55300 QT$=".SE":GOSUB58500
55320 IFQN%=Q3THEN55100
55340 QT=Q1:GOTO55300
55400 QT=Q2:IFQT=0THENRETURN
55420 QT$=".EX"
55440 GOSUB58500
55460 IFQ1<>0THENQT=Q1:GOTO55440
55480 RH%=QX/256:RL%=QX-256*RH%
55500 R$=CHR$(RH%)+CHR$(RL%)
55520 GOSUB63000:QX=Q2
55540 R=1:R$=Q2$:GOSUB63000
55560 RETURN

READY.
```

A szubrutin leírása:

55020–55040 A QN változóból képezzük az elsődleges file rekordsorszámát (QT).

55060–55080 Beolvassuk az elsődleges file-ból a kódmezőt. Ha QN% ≠ 0, a rekord a másodlagos file-ban található.

55100 Ha a rekord üres (Q4 = 0), visszatérünk a főprogramba.

55120–55140 A rekordot töröljük, de a láncolt címeket még tárolni kell, mert előfordulhatnak még folytató rekordok. Ha az elsődleges file-ban töröltünk, nem kell visszaláncolást végezni (QT$ = ".pr").

55150–55160 Az előző lépésben beolvasott rekordot ismét beolvassuk és beleírjuk a törlendő rekord láncolási paramétereit, hogy a törlés ne rontsa el a láncolást (lásd az előbbi táblázatokat is.)

55180–55260 Az eddigi első szabad rekord sorszámát beírjuk a törölt rekordba és az utóbbi sorszáma adja az új első szabad sorszámot, amit a szélső rekordban és a QS változóban tárolunk.

55300–55340 A másodlagos file-t a láncolások alapján addig vizsgáljuk, amíg a keresett rekordot meg nem találjuk. (Q% = Q3)

55400 Ha nincs megjegyzés (Q2 = 0), akkor visszatérünk a főprogramba, egyébként a törlést folytatjuk.

55420–55560 A megjegyzéseket nem fizikailag töröljük. A lánc első tagja lesz az első szabad rekord, az utolsóba pedig beírjuk a régi szabad sorszámot. Az első rekordot és a QX változót megfelelően módosítjuk.

2.3.1.4.4 Egy rekord felírása a file-ba

A rekordok törlésének logikáját megértve könnyen követheti az Olvasó az adatbevitelhez fűzött magyarázatot. Az adatbevitel ugyan az adatkezelés műveleti sorrendje szerint az előző

fejezet elé kívánkozott volna, mi azonban azon a véleményen vagyunk, hogy logikailag ide illeszkedik jobban.
A szubrutin meghívása előtt egyetlen változónak kell értéket adni. A QD$ tartalmazza a felírandó rekordot. A többi változó ismert.

A szubrutin listája:

```
P15

57000 :
57001 REM **** A TORZSREKORDOK BEVITELE ****
57002 :
57020 GOSUB58800:QN%=0
57040 QT$=".PR":QT=HC:GOSUB58500
57060 IFQ4<>0THENQT$=".SE":GOTO57260
57080 RC%=255:GOSUB63500
57100 R$=LEFT$(R$,5)+LEFT$(QD$,QL%):GOSUB63000
57200 RETURN
57260 IFQ3=14THENQN%=13
57280 IFQ3>QN%+1THENQ6=QT:DN$=QN$+QZ$:R=QZ:GOTO57330
57310 QN%=Q3:IFQ1<>0THENQT=Q1:GOSUB58500:GOTO57260
57320 Q6=0
57330 RC%=255:GOSUB65300
57340 RH%=QS/256:RL%=QS-256*RH%
57360 R$=CHR$(RH%)+CHR$(RL%)+MID$(R$,3,QL%+3):GOSUB63000
57400 QT=QS:GOSUB58500
57410 QN%=QN%+1:IFQN%=13THENQN%=QN%+1
57420 IFVAL(F1$)<>0THEN57440
57430 IFQ2<>0THENQS=Q2:GOTO57480
57440 QS=QS+1:RH%=QS/256:RL%=QS-RH%*256
57450 IFRH%=13THENQS=QS+256
57460 IFRL%=13THENQS=QS+1
57480 RH%=Q6/256:RL%=Q6-256*RH%
57500 R$=CHR$(RH%)+CHR$(RL%)+CHR$(0)+CHR$(0)
57510 R$=R$+CHR$(QN%)+LEFT$(QD$,QL%)
57520 GOSUB63000
57540 RH%=QS/256:RL%=QS-256*RH%
57560 R$=CHR$(RH%)+CHR$(RL%):R=1:GOSUB63000
57580 RETURN

READY.
```

A szubrutin leírása:

57020–57200 A rekordszám kiszámítása után – ha az elsődleges file-ban a megfelelő hely szabad (Q4 = 0), az új rekordot erre a helyre, egyébként a másodlagos file-ban keresett szabad területre írjuk.
57260–57310 Megkeressük a másodlagos file első szabad sorszámát a láncolás alapján.
57330–57360 A talált szabad rekordba folytató sorszámként beírjuk a QS változó értékét.
57400–57520 Az új rekordot felírjuk a file-ba. Attól függően, hogy az üres sorszám egy törölt rekord volt-e (F1$ = "00"), vagy új rekord a file végéről, a QS változó értékét módosítjuk.
57540–57580 Az első rekord tartalmát aktualizáljuk.

A teljes adatbevitelből még hiányzik a megjegyzés bevitele. A megjegyzést ugyanúgy tárolhatjuk, mint magát a rekordot, annyi módosítással, hogy a rutin hívása előtt a QX$ változóban a megjegyzés szövegét kell elhelyezni.

Az egyébként még szükséges paramétereket az előzőleg feldolgozott törzsrekordból vesszük. Ha egy törzsrekordhoz a megjegyzést csak egy későbbi időpontban akarjuk hozzáfűzni, a törzsrekordot előzőleg be kell olvasnunk és a QT és QT$ paramétereket megfelelően kell módosítanunk.

A megjegyzést felvivő szubrutin listája:

```
P16

56000 :
56001 REM **** A MEGJEGYZESEK BEOLVASASA ****
56002 :
56010 QF%=0:IFQT$=".EX"THENQF%=1:RETURN
56020 GOSUB58500:Q3=0
56040 QT$=".EX":IFQ2<>0THEN56200
56060 RC%=255:GOSUB63500
56080 RH%=QX/256:RL%=QX-256*RH%
56100 R$=LEFT$(R$,2)+CHR$(RH%)+CHR$(RL%)+MID$(R$,5,QL%+1)
56120 GOSUB63000:Q3=1:GOTO56260
56140 IFQ2=0THENQ2=QT:GOTO56260
56200 Q1=Q2:QT=Q2:GOSUB58500
56220 Q2=Q1:GOTO56140
56260 QT=QX:GOSUB58500
56280 Q4:QX:IFVAL(F1$)<>0THEN56300
56290 IFQ1<>0THENQX=Q1:GOTO56340
56300 QX=QX+1:RH%=QX/256:RL%=QX-256*RH%
56310 IFRH%=13THENQX=QX+256
56320 IFRL%=13THENQX=QX+1
56340 R$=CHR$(0)+CHR$(0)+LEFT$(QX$,QX%):GOSUB63000
56360 ONQ3GOTO56500
56380 R=Q2:RC%=255:GOSUB63500
56400 RH%=Q4/256:RL%=Q4-256*RH%
56420 R$=CHR$(RH%)+CHR$(RL%)+MID$(R$,3,QX%+1)
56440 GOSUB63000
56500 RH%=QX/256:RL%=QX-256*RH%
56540 R=1:R$=CHR$(RH%)+CHR$(RL%):GOSUB63000
56560 RETURN
56600 Q2=QT:Q3=0:QF%=0
56620 IFQT$<>".EX"THENQF%=1:RETURN
56640 GOTO56260

READY.
```

56020–56120 Ha még nincs megjegyzés (Q2 = 0), akkor a láncolási címet (QX) a törzsrekordba írjuk.

56140–56220 Megvizsgáljuk a rekordhoz tartozó megjegyzések láncolatát, hogy az új megjegyzést közéjük illesszük.

56260–56340 A megjegyzést beírjuk a file-ba, és a QX-nek új értéket adunk aszerint, hogy bővítettük-e a file-t, vagy korábban felszabadított rekordba írtunk.

56360–56440 Ha már vannak megjegyzések (Q3 = 0), akkor az utolsóba be kell írni az új rekord sorszámát.

56500–56560 Az első rekord tartalmát aktualizáljuk.

56600 Beugrás a rutinba arra az esetre, ha ugyanahhoz a törzsrekordhoz további megjegyzéseket kell fűzni. A láncolási paraméter még megvan, ezért csak annyit kell tenni, hogy az új megjegyzést beírjuk a file-ba. A szubrutint ezen a ponton csak akkor hívhatjuk, ha közvetlenül előtte írtunk egy megjegyzést a rekordba.

A programcsomagból már csak a kereső rutin hiányzik.

2.3.1.4.5 Egy rekord keresése a QUISAM file-ban

A törzsrekord és a megjegyzés kereséséhez egyaránt két paraméterre van szükség: a HASH kódra és a teljes azonosítóra.
A megjegyzéseket mindig a törzsrekordon keresztül kereshetjük meg, hiszen a törzsrekordban van az első megjegyzés rekordszáma.
Foglalkozzunk tehát először a törzsrekord keresésével.
A keresőkritériumot, például az utónevet a QK$ változóban át kell adni a szubrutinnak, minden egyébről a rutin gondoskodik. A szubrutin a keresett rekordon kívül (QD$) a megtalált rekord sorszámát is visszaadja arra az esetre, ha a későbbi keresés meggyorsításához tárolni akarjuk. A keresés eredményességét a QF% változó értéke jelzi. Ha QF% ≠ 0, akkor a keresés sikertelen volt.

A szubrutin listája:

```
P17

54000 :
54001 REM **** KULCS SZERINTI KERESES ****
54002 :
54020 HC$=LEFT$(QK$+"      ",5)
54040 GOSUB58840
54060 QT$=".PR":QT=HC:QF%=0
54080 GOSUB58500
54100 RC%=255:GOSUB63500
54120 IFMID$(R$,QK%+5,LEN(QK$))=QK$THEN54200
54140 IFQ1=0THENQF%=1:RETURN
54160 QT=Q1:QT$=".SE":GOTO54080
54200 QN=HC*100+Q3
54220 QD$=MID$(R$,6,QL%)
54240 RETURN

READY.
```

A szubrutin leírása:

54020–54060 A keresést az elsődleges file-ban kezdjük, a QK$-hoz rendelt rekordsorszám (HASH kód) alapján.

54100–54160 Beolvassuk a rekordot az elsődleges file-ból, a rekord QK% számú karakterét összehasonlítjuk a QK$ tartalmával, ha nem egyezik és van folytató rekord, akkor folytatjuk az olvasást a másodlagos file-ban.

54200–54240 Kiszámítjuk a megtalált rekord azonosító számát, az R$ változóból kiemeljük a hasznos adatokat és a QD$ változóban átadjuk a főprogramnak.

A fent leírt eljáráshoz hasonló az azonosító szerinti keresés. A keresett azonosítót a QN változóban adjuk át a rutinnak. Az azonosító alapján csak a kódmezőt olvassuk be és megvizsgáljuk, hogy megegyezik-e azzal a sorszámmal, amit előbb a QN-ből levágtunk (QN% = Q3). A teljes rekordot ismét átadjuk a főprogramnak.

A szubrutin listája:

```
P18

54500 :
54501 REM **** SORSZAM SZERINTI KERESES ****
54502 :
54520 QN%=VAL(RIGHT$(STR$(QN),2))
54540 QT=(QN-QN%)/100:QF%=0
54560 QT$=".PR":GOSUB58500
54580 IFQN%=0THEN54700
54590 IFQ1=0THENQF%=1:RETURN
54600 QT$=".SE"
54620 QT=Q1:GOSUB58500
54640 IFQ3=QN%THEN54700
54660 IFQ1=0THENQF%=1:RETURN
54680 GOTO54620
54700 IFQ4=0THENQF%=1:RETURN
54710 RC%=255:GOSUB63500
54720 GOTO54220

READY.
```

A megjegyzések különböző szempontok szerinti keresését már összefoglaltuk, hiszen a két előző rutinból építhető fel. Az egyezés megállapítása után a láncolási cím (Q2) alapján megtalált első megjegyzést átadjuk a főprogramnak.
A szubrutin 53200-as sorában vizsgáljuk meg, hogy van-e folytató rekord. A keresés eredménytelenségét most is a QF% < >0 reláció jelzi.

A szubrutin listája:

```
P19

53000 :
53001 REM **** A REKORD KERESESE KULCS SZERINT
53002 :
53020 QF%=0:GOSUB54000
53040 GOTO53200
53100 :
53101 REM **** A REKORD KERESESE SORSZAM SZERINT ****
53102 :
53120 QF%=0:GOSUB54500
53200 :
53201 REM **** A KOVETKEZO BEVITEL KERESESE ****
53202 :
53220 IFQF%THENRETURN
53240 IFQ2=0THENQF%=2:RETURN
53260 R=Q2:DN$=QN$+".EX":RC%=255
53280 GOSUB 63500
53300 QX$=MID$(R$,3,QX%)
53320 Q2=ASC(LEFT$(R$,1))*256+ASC(MID$(R$,2,1))
53340 RETURN

READY.
```

2.3.1.4.6 Mire alkalmas a QUISAM file?

Azt, hogy egy feladat megoldására célszerű-e a QUISAM file-szervezés vagy nem, mindig a tárolandó adathalmaz mérete, és az adatok jellege alapján kell eldönteni.
Az adatok elérési sebessége nagymértékben függ az elsődleges file méretétől.
Tegyük fel, hogy egy pénztárnapló adatait akarjuk lemezen tárolni. A bevételeket és a kiadásokat folyamatosan kell könyvelni. Mivel a VC 1541-es lemezegységen egyszerre csak egy relatív file lehet nyitva, a file gyakori nyitása és zárása pedig igen sok időt vesz igénybe, célszerű a bevételeket és a kiadásokat a megjegyzés file-ba könyvelni. Legyen a keresési mező a napi dátum. A napi adatok feldolgozása során a kód nem változik, az elsődleges file mérete kicsi és a másodlagos file-ra is ritkán lesz szükség.
A folyamatos adatbevitel és az adatok visszaolvasása ezzel a file-szervezéssel optimálisan megoldható.
Gyökeresen más megoldást kell választanunk a címnyilvántartáshoz. A címben szereplő vezetéknevek ugyanis rendkívül változatosak lehetnek, a hozzájuk rendelt HASH kódok tehát gyakran fognak változni. Az előző feladattal szemben most az a célszerű, ha az elsődleges file-t lehetőleg nagyra méretezzük.
Ugyanis, ha az elsődleges file viszonylag nagy, a kódok változása nem kényszeríti a rendszert arra, hogy gyakran forduljon a másodlagos file-hoz és emiatt az elsődleges file-t le kelljen zárnia.

Eljutottunk a QUISAM file-szervezéssel kapcsolatos tanulmányaink végére. Az összegyűlt szubrutinok láttán felmerülhet az Olvasóban a kérdés, hogy mennyi hely marad a tárban a saját programja számára, ha a szubrutinok mindegyikét használni szeretné. A szubrutinok összesen 15 kbyte-ot foglalnak el, ami valóban nem kevés. Mi az, ami mégis megnyugtatásul szolgálhat? A könyv végén közlünk egy programot, amit az irodalomjegyzékek nyilvántartására készítettünk QUISAM file-szervezéssel. Ha az Olvasó átnézi a programlistát, azonnal látja, hogy a segédrutinokhoz nagyon kevés kiegészítésre volt szükség. Néhány főprogramsorral megvalósítottuk a teljes adatkezelést (beleértve a képmaszk elkészítését is).
Kis gyakorlat után az újabb és újabb adatbázis-kezelő rendszerek megírása nem jelent több munkát, mint a szubrutinok összefűzése. A rutinok nagyon fontos tulajdonsága, hogy mind azonos névvel hivatkoznak az azonos változókra.
Bízunk benne, hogy a fejezet végére érve az Olvasó úgy érzi, hogy az eredmény megérte a fáradságot.
Még egy fontos megjegyzés:
A QUISAM a DATA BECKER cég Fejlesztési Osztály vezetője, Klaus Gerits által kidolgozott új módszer. A legnagyobb precizitás ellenére előfordulhatnak benne hibák. Az Olvasó minden javaslatát és építő bírálatát (írásos formában) a szerző szívesen fogadja.

2.3.2 A nyomtatás

A legtöbb számítógépes adatfeldolgozás nyomtatással végződik. „Az eredmények feketén fehéren..." – ez a fontos a program használója számára.
A dolog másik oldala – a nyomtatás programozása – már korántsem akkora öröm, mint az eredmények feketén fehéren.
A nyomtatás egyrészt az egyik legegyszerűbb és egyben legunalmasabb, másrészt az egyik legtöbb bosszúságot okozó, legalattomosabb programozói feladat.
Addig általában nincs különösebb gond, amíg a saját jól ismert nyomtatónkkal dolgozunk.

A különböző típusú nyomtatók programozása azonban jelentősen eltér egymástól.
Mi a teendő akkor, ha nem tudjuk előre, hogy a programnak milyen nyomtatóval kell együttműködnie?

2.3.2.1 Nyomtató = Nyomtató?

Ismeri az Olvasó azt a bizsergést, ami az ember hátában kezdődik, azután fokozatosan felkúszik a feje búbjáig és az arcán egyfajta elszíneződést hagy hátra, aminek láttán a gyerekeknél rossz lelkiismeretre tippelünk? Ha nem ismerné, nézze meg egyszer azt a szoftverkészítőt, aki éppen arra válaszol, hogy vajon az X program az Y nyomtatóval is képes-e együtt dolgozni!
Mi a jelenség magyarázata?
Vegyes hardver esetén (egy rendszerben több különböző gyártótól származó készülék) eleve nem biztos, hogy az egyes egységek – a fizikai csatlakoztatási lehetőségektől eltekintve – illeszthetőek egymáshoz. A Centronics nyomtató például nem kifejezetten a C 64-eshez készült. A C 64-es esetében a kérdés rögtön ott kezdődik, hogy egy nyomtató ismeri-e egyáltalán a grafikus jeleket?
Az idegen nyomtatók illesztési gondjainál sokkal súlyosabb azonban az a probléma, hogy a C 64-eshez tervezett Commodore nyomtatókat sem lehet programtechnikailag egyformán kezelni.
Hogy egy konkrét példát említsünk:
Tegyük fel, hogy készítettünk egy programot, amely az eredményeket SEIKO GP–100VC nyomtatóra írja. A nyomtatás során nagy- és kisbetűkre egyaránt szükségünk van, de lusták voltunk minden sor elé kiírni a CHR$(17) karaktert, ami a nagybetűs/kisbetűs üzemmód beállításának karaktere. A nyomtatót egyszerűen az OPEN1,4,7 utasítással nyitottuk meg, ami ennél a nyomtatónál azt jelenti, hogy az adatokat (a 7-es másodlagos címmel jelölt adatcsatornán) nagybetűs/kisbetűs ábrázolásban kell kinyomtatni.
Ugyanez a program a VC 1526-os nyomtatóval nagy szitkozódással kísért hibakereséshez vezet, a nyomtató ugyanis meg se moccan. A dolog egyszerű oka az, hogy a VC 1526-os nyomtató a 7-es másodlagos című csatornán nem fogad el adatokat, mert a 7-es cím nála az üzemmód átkapcsolási címe. A kívánt hatás eléréséhez be kell iktatnunk még néhány utasítást:

```
10 OPEN1,4,7:PRINT1:CLOSE1:REM        átkapcsolás nagy/kisbetűre
20 OPEN1,4

30 PRINT#1,"adatok"
```

Egy másik példa: a VC 1526-os nyomtató egy büszke tulajdonosa az 1-es és 2-es másodlagos címekkel vezérli a nyomtatás formátumát. Programja sohasem fog működni a GP–100VC nyomtatóval, mert az utóbbi nem úgy értelmezi a másodlagos címeket, mint a VC 1526-os. A nagybetűs/kisbetűs üzemmód vezérlésére célszerű CHR$(17) karaktert használni, mert ezt a lehetőséget a legtöbb nyomtató ismeri. Más a helyzet az adatok tabulálásával. A tabulálásra nagyon kevés nyomtató képes, a VC jelzésű nyomtatók közül pl. csak a VC 1526-os. Pedig a tabulálás jelentős mértékben megkönnyíti a programozást és ugyanakkor nagymértékben javítja a nyomtatott lap megjelenési formáját. Érdemes ezzel a kérdéssel egy kicsit részletesebben foglalkozni, és átgondolni, hogy miként lehetne a tabulálást legalább programtechnikailag szimulálni, ha már a nyomtatók többsége magától nem képes erre. Pontosan ez a tárgya

a következő fejezeteknek. Hogyan készíthetünk megfelelő nyomtatási képet anélkül, hogy intelligens nyomtatót feltételeznénk?
Egy feladatot minden nyomtató hiba nélkül elvégez: olyan sorrendben nyomtatja a karaktereket, amilyen sorrendben azok hozzá érkeznek.
A fentiekből következik, hogy csak akkor kaphatunk a nyomtató típusától függetlenül minden esetben a tervezettnek megfelelő nyomtatásképet, ha az adatokat a nyomtatáshoz előkészítjük.
A feladat nagyon egyszerű, ha pl. egy levelet kell kinyomtatnunk. Kevésbé megnyugtató azonban egy számla nyomtatásának gondolata. A számlán minden adatnak előre pontosan meghatározott helye van. A szöveget és a számokat pozicionálni kell, a szöveget rendszerint balra, a számokat jobbra tömörítve.
A következő fejezetben összegyűjtjük a nyomtatás előkészítését megkönnyítő programozási eszközöket.

2.3.2.2 A tabulátor

A megjelenési kép vezérlésére szolgál a BASIC TAB függvény. Egyetlen hibája, hogy másként működik íratásnál ill. nyomtatásnál. Valószínűleg feltűnt már az Olvasónak is, hogy például a

```
PRINT"HALLO"TAB(10)"HOGY MEGY"
```

utasítás eredménye a képernyőn:

```
HALLO          HOGY MEGY
```

a nyomtatón azonban:

```
HALLO              HOGY MEGY
```

Hogyan lehetséges ez? A magyarázat: az operációs rendszer a TAB(X) utasítás helyett SPC(X) utasítást hajt végre, ha a kiírás nem a képernyőre vonatkozik.
A TAB függvény a kurzort a képernyő szélétől számítva annyi hellyel mozdítja jobbra, amennyi az argumentuma, ezzel szemben az SPC a pozíciót a nyomtatófej pillanatnyi állásához viszonyítja.
Az SPC utasítás hatására az első szöveg hosszúságától függően a második szöveg eleje eltolódhat.
Egy megoldási javaslat Commodore nyomtatókra, tabuláláshoz: Kinyomtatjuk a sor első részét az első oszloptól kezdve, majd soremelés nélkül visszahúzzuk a nyomtatófejet a sor elejére (soremelés nélküli kocsivissza), majd az SPC függvénnyel a kívánt pozícióra állunk.
A soremelés nélküli kocsivissza karakterkódja: CHR$(141). A számla egy sorát a következő BASIC sorral nyomtathatjuk ki tabulálva:

```
10 OPEN1,4
20 PRINT#1, "HOSSZABITO ZSINOR"CHR$(141)SPC(40)"145 FT"
```

A megnevezést a papír bal oldalára tömörítve, az egységárat pedig a 40. oszloptól kezdve nyomtatjuk. A megoldás helyettesíti a TAB függvényt a nyomtatásnál. A számlakép azonban

még mindig nem felel meg a követelményeknek. A számok ugyanis mindig a 40. oszlopon kezdődnek, függetlenül a nagyságuktól. Nem valószínű, hogy a számla címzettje az eredménytől el lesz ragadtatva.

2.3.2.3 A tizedespontok

Tegyük fel, hogy a számlán szereplő cikkek értékének összege: 852.40 Ft. A fenti BASIC sorokkal kinyomtatva, az összeg a 40. oszlopon kezdődik, de a tizedespont az előző sorok tizedespontjához képest egy oszloppal jobbra tolódik. Eltérő nagyságrendű számok esetén a tizedespontok sohasem kerülnének egymás alá. Hogyan lehet a számlát a szabványok szerint kinyomtatni úgy, hogy a tizedespontok egymás alá kerüljenek?
Például úgy, hogy a számot füzérben tároljuk, megvizsgáljuk, hogy a tizedespont a szövegben hányadik helyen van és az SPC függvény argumentumát ettől tesszük függővé.
Az eljárás kicsit körülményes. Nem javasoljuk, hogy az Olvasó nekilásson a programozásának!
Miért?

2.3.2.4 A nyomtatáskép formátuma

A nyomtatás formátuma alatt egy kinyomtatott sor megjelenési képét értjük. Milyen egyszerű lenne a helyzet, ha a formátumot programból vezérelhetnénk. Megadhatnánk például, hogy a számla adatsorát milyen numerikus és alfanumerikus adatmezők alkotják, és ezeknek milyen legyen a formátuma.
A nyomtatáskép formátumának vezérlésére készítettük az alábbi szubrutint:

```
P20

48000 :
48001 REM **** FORMAZOTT ADATKIIRAS ****
48002 :
48020 IFFZ%=0THENFZ%=1:FL%=LEN(FO$):FP$=""
48040 FI%=LEN(FI$):FB$="":FC%=1
48050 IFMID$(FO$,FZ%,1)=" "THENFZ%=FZ%+1:FB$=FB$+" ":GOTO48050
48060 FM$=MID$(FO$,FZ%,1)
48080 FV%=0:FH%=0:FA%=0:FB%=0:FF%=0
48100 IFMID$(FO$,FZ%,1)="."THENFZ%=FZ%+1:FF%=2:GOTO48180
48120 IFFZ%>FL%THENFA%=FA%+1:FZ%=FZ%+1:GOTO48240
48140 IFMID$(FO$,FZ%,1)=" "THEN48240
48160 FA%=FA%+1:FZ%=FZ%+1:GOTO48100
48180 IFMID$(FO$,FZ%,1)=" "ORFZ%>FL%THEN48240
48200 FB%=FB%+1:FZ%=FZ%+1
48220 GOTO 48180
48240 IFMID$(FO$,FZ%,1)="."THENFC%=FC%+1:FF%=FF%+4:GOTO48320
48260 FV%=FV%+1:FC%=FC%+1
48280 IFFC%>FI%THEN48380
48300 GOTO48240
48320 IFFC%>FI%THEN48380
48340 FH%=FH%+1:FC%=FC%+1
48360 GOTO 48320
48380 IFFM$="A"THEN48780
48400 L2=FA%-FV%:IFL2<0THENFF%=FF%+1:GOTO48720
48420 IFL2=0THEN48460
48440 FORL1=1TOL2:FB$=FB$+" ":NEXTL1
48460 IFFV%=0THEN48520
```

```
48480 L2=FV%:FORL1=1TOL2
48500 FB$=FB$+MID$(FI$,L1,1):NEXTL1
48520 IFFF%=0THEN48920
48540 FF%=0:FB$=FB$+".":L2=FV%+2
48560 IFFB%=0THEN48920
48580 IFFB%=FH%THENL3=FB%:GOTO48620
48600 L3=-FH%*(FB%>FH%)-FB%*(FH%>FB%)-1
48620 FORL1=L2TOL2+L3
48640 FB$=FB$+MID$(FI$,L1,1):NEXTL1
48660 IFFH%>=FB%THEN48910
48680 L2=FB%-FH%:FORL1=1TOL2
48700 FB$=FB$+"0":NEXTL1:GOTO48910
48720 L2=FA%+FB%:IF(FF%AND2)THENFF%=1:L2=L2+1
48740 FORL1=1TOL2:FB$=FB$+"*"
48760 NEXTL1:GOTO48910
48780 IF(FF%AND4)THENFV%=FV%+FH%+1
48800 L2=FV%:FORL1=1TOL2
48820 IFMID$(FI$,L1,1)<>" "THENL3=L1:L1=L2
48840 NEXTL1
48860 L2=-FV%*(FA%>FV%)-FA%*(FV%>FA%)
48870 IFL2=0THENL2=FV%
48880 FORL1=L3TOL3+L2
48900 FB$=FB$+MID$(FI$,L1,1):NEXTL1
48902 IFFV%>FA%THEN48910
48904 L3=FA%-L2:FORL1=1TOL3
48906 FB$=FB$+" ":NEXTL1
48910 IFFZ%>FL%THENFZ%=0:GOTO48960
48920 L2=FZ%:L3=FL%:FORL1=L2TOL3
48930 IFMID$(FO$,L1,1)<>" "THENFZ%=L1:L1=L3:GOTO48950
48940 FB$=FB$+" "
48950 NEXTL1
48960 FP$=FP$+FB$:FF%=(FF%AND1)
48970 RETURN

READY.
```

A szubrutinban használt változók:

FO$	A kívánt sorformátum
FI$	Az előkészítendő adat
FB$	Az éppen elkészített részfüzér
FP$	A teljes nyomtatási füzér
FZ%	Mutató, amely az éppen feldolgozott rész elejére mutat FO$-ban.
FF%	A tizedesvessző előtti helyek átfutását jelzi (FF% < >0)
FA%	Az FO$-ban éppen feldolgozott mező tizedesvessző előtti helyeinek száma
FB%	Mint fent, de a tizedesvessző utáni helyek
FV%	Az FI$-ban lévő tizedesvessző előtti helyek száma
FH%	Az FI$-ban levő tizedesvessző utáni helyek száma

A szubrutin leírása:

48020–48080 Ha nem először hívták meg a rutint (FZ% = 0), néhány változó eredeti értékét visszaállítjuk.

48100–48160 Az éppen feldolgozott mező tizedesvessző előtti helyeinek számát tároljuk az FA%-ban.

48180–48220 Mint fent, a tizedesvessző utáni helyekre (FB%).

48240–48300 Az FI$-ban lévő tizedesvessző előtti helyek számát tároljuk az FV%-ban.

48320–48360 Az FI$-ban lévő tizedesvessző utáni helyek számát tároljuk az FH%-ban.

48400–48440 A numerikus mező összeállítása. Ha az FI$ változó FO$-nál kevesebb tizedesvessző előtti helyet tartalmaz, akkor azokat üres karakterekkel töltjük ki.
48460–48500 Az érvényes tizedesvessző előtti karakterek az FB$-ba kerülnek.
48560–48640 Az érvényes tizedesvessző utáni helyek az FB$-ba kerülnek.
48660–48700 Ha az FI$ változó FO$-nál kevesebb tizedesvessző utáni helyet tartalmaz, akkor azokat nullákkal töltjük ki.
48720–48760 Ha az FI$ változó FO$-nál több tizedesvessző előtti helyet tartalmaz, akkor a teljes mezőt *-gal töltjük ki, hogy a nyomtatásnál szembetűnjék. A nyomtatási mezőt kicsire méreteztük. A program ezt az állapotot FF% < >0-val jelzi.
48780–48840 Itt kezdődik az alfanumerikus mezők előkészítése. Először a bevezető üres helyeket elhagyjuk az FI$-ból, hogy a látható karakterek valóban a mező elején kezdődjenek.
48860–48900 Az FI$-ból a karaktereket az FO$ szerint az FB$-ba visszük át. Az FI$ többi karakterét levágjuk, az FO$ többi karakterét pedig üres helyekkel töltjük ki.
48910–48950 Az FO$ változóban a következő mezőtől való távolságnak megfelelően az FB$-t üres helyekkel töltjük ki.
48960–48970 Az FB$-t hozzáadjuk az FP$-hoz.

A szubrutin kezelése rendkívül egyszerű:
Állapítsuk meg, hogy hány mező található a sorban és hogy ezek alfanumerikus adatokat vagy számokat tartalmaznak.
Rögzítsük ezt a formátumot az FO$-ban. A formátum a számla esetében a következőképpen nézne ki:

FO$="99 999 aaaaaaaaaa 999.99 9999.99"

A formátumban a numerikus mezőket 9-cel jelöljük, az alfanumerikus mezőket pedig a-val. Az eltérő kezelésnek az a magyarázata, hogy a program az adatokat az alfanumerikus mezőkben balra tömörítve adja vissza, a numerikusokat azonban a saját és az FO$-ban található tizedespont-helyzet egyeztetésével jobbra tömörítve. Az FI$ változóban kell megadni a nyomtatandó adatokat, és annyiszor kell a rutint meghívni, ahány mezőt meghatároztunk az FO$ változóban, természetesen úgy, hogy az FI$ változó mindig más adatot tartalmazzon. Az eljárás végén a teljes nyomtatási sor az FP$-ban áll rendelkezésre.

A számla egy sorát a következő kis programmal lehet összeállítani:

```
P21

10 OPEN1,4
20 FO$="99  999  AAAAAAAAAA  999.99  9999.99"
30 FI$=STR$(PO):GOSUB48000:REM POZICIOSZAM
40 FI$=STR$(AN):GOSUB48000:REM SORSZAM
50 FI$=AR$:GOSUB48000:REM CIKKSZAM
60 FI$=STR$(EP):GOSUB48000:REM EGYSEGAR
70 FI$=STR$(AN*EP):GOSUB48000:REM ERTEK
80 PRINT#1,FP$

READY.
```

A programból a rutint ötször hívtuk meg, vagyis annyiszor, ahány mezőt az FO$-ban meghatároztunk.

Az FO$-ban lévő formátumot természetesen csak egyszer kell átadni. Ez mindaddig érvényben marad, amíg a rutin hívása előtt tartalmát felül nem írjuk.
A rutin nagy segítséget nyújt a címkenyomtatásnál is. Az öntapadó címkék nyomtatására általában olyan papírt használnak, ahol a hordozón a címkék egymás alá vannak felragasztva. Befűzésnél a címke a nyomtató bal oldalára kerül és így a címkéket normál PRINT utasításokkal kitölthetjük. Nagy könnyebbséget jelent a nyomtatásnál az előző rutin, hiszen a szöveg hosszát megszabva elkerülhetjük, hogy túlszaladjunk a címkén. Mivel azonban a cím általában rövid sorokból áll, amelyeket egymás alá kell nyomtatni, a nyomtatófejet a címke minden sorához egyszer el kell mozdítani, majd új sort kell kezdeni. Ez mind időbe telik. Sokkal gyorsabban megy a nyomtatás, ha a nyomtató maximális sorhosszúságának megfelelően választjuk meg a címkelapot és ha a címkék kettesével vagy hármasával vannak egymás mellé felragasztva. Ekkor ugyan a programozó számára a nyomtatáselőkészítés lesz rendkívül bonyolult, mert eltekintve attól, hogy két-két (vagy három-három) címke sorának adatait kell előkészíteni, elkerülhetetlen, hogy a nyomtatásképet abban a sorrendben állítsa össze, ahogyan a nyomtatás megköveteli.
Három megoldás közül választhatunk:
Ha a nyomtató tud tabulálni, akkor előre megválaszthatjuk a tabulátor helyzeteit és a nyomtatás nem jelent gondot. Sajnos a Commodore nyomtatók nem rendelkeznek ezzel a sajátossággal. A 2.3.2.1 pontban ismertetett eljárás is használható. Ez ugyan valamivel körülményesebb, de működik.
A harmadik és legelegánsabb lehetőség a formátumozó rutin használata. A címkék elfoglalt helyzetének megfelelően be kell állítani az FO$ változó értéket, és ezután minden megy magától. Kétoszlopos címkelap esetén a program a következőképpen nézne ki:

```
100 FO$="AAAAAAAAAAAAAAAAAAAA      AAAAAAAAAAAAAAAAAAAA"
 :
 :
200 A1$="MEGSZOLITAS1":A2$="MEGSZOLITAS2"
210 FI$=A1$ : GOSUB 48000 : FI$=A2$ : GOSUB 48000
220 PRINT#1,FP$
```

Hasonlóan folytathatjuk a címke többi sorának kitöltését is. Az előny kézenfekvő: bárhogy vannak elrendezve a címkék, csak az FO$ változó tartalmát kell megfelelően módosítani és a nyomtató egy sorhúzással egyszerre több címkét készít.

Ezzel a második fejezet végéhez érkeztünk és reméljük, hogy a programozás mindhárom fázisát sikerült érthető formában megmagyaráznunk. Célkitűzésünk volt, hogy hasznos eszközöket adjunk az Olvasó kezébe, és az adott összefüggésben megvitassuk minden témakörnél az alapokat, egyben saját kísérletekre bátorítva az Olvasót. Természetesen tisztában vagyunk azzal, hogy egyes programrészeket gyorsabban és előnyösebben oldhattunk volna meg gépi kódban, ezzel azonban sokukat kizártuk volna az olvasók táborából.

Örömmel fogadjuk, ha Olvasóink tájékoztatnak bennünket azokról a programozási ötletekről, amik a könyv tanulmányozása során jutottak eszükbe. Mi is szívesen tanulunk bárkitől.

3. FEJEZET
ÖTLETEK A MAGASSZINTŰ PROGRAMOZÁSTECHNIKÁHOZ

Ebben a fejezetben témakörönként csoportosítva bemutatjuk, hogyan írhatunk jól használható és könnyen kezelhető programokat.

3.1 A programok paraméterezése

Képzeljük el, hogy elkészültünk egy programrendszerrel, amely bizonyos adathalmaz (pl. címek) tárolására, karbantartására szolgál. Időközben azonban kiderült, hogy az egyik adatmező (pl. a vezetéknév) hosszát rövidre szabtuk, emiatt módosítanunk kell a programokat. Minden programban meg kell keresnünk azt vagy azokat a programsorokat, ahol az adatmező hosszúságára hivatkoztunk. Ráadásul ha a rekordon belül ez az adatmező nem az utolsó helyen volt, akkor megváltozik az összes többi adatmező rekordon belüli kezdőpozíciója is. Előreláthatóan nem kis munkát okoz ez a látszólag apró javítás. Hogyan lehet elkerülni az ilyen problémákat?
A megoldás a program paraméterezése. A paraméterezés azt jelenti, hogy a programban bizonyos (pl. az adathalmaz szerkezetét meghatározó) adatokat nem konstansként, hanem változóként adunk meg. A program így sokkal általánosabb lesz. A kiemelt változók értékét minden speciális feladatban rögtön a program elején megszabhatjuk, ez az ún. inicializálás. Az inicializálás lehet néhány egyszerű értékadó utasítás, vagy beolvasás DATA sorokból.
Nézzük meg az alábbi példát, amely egy rekordot több változóból épít fel:

```
100 D$=A1$+A2$+A3$+A4$
110 PRINT#1,D$
```

Ugyanez paraméterezve:

```
P22

100 GOSUB 1000:REM INICIALIZALAS
110 D$=""
120 FOR I=1 TO N:REM A VALTOZOK SZAMA
130 D$= D$+ A$(I) :NEXT
140 PRINT#1, D$
150 :
160 :
1000 N=5:DIM A$(N):RETURN

READY.
```

Első pillantásra a második megoldás körülményesebbnek tűnik. Valóban az is, de megvan az az óriási előnye, hogy a változók számának bővülése esetén csak az N értékét kell megváltoztatni az inicializáló szubrutinban.

Milyen értékeket célszerű paraméterezni?
A címek nyilvántartási rendszerében például a rekordon belüli mezők számát és hosszúságát. Ezekből a program ki tudja számítani a teljes rekord hosszúságát. Inicializáláskor az egyes adatmezők megnevezését lerakhatjuk DATA sorokba:

```
P23

100 READ N: REM AZ ADATMEZOK SZAMA
110 DIM P(N+1), L(N): REM POZICIO, HOSSZ
120 P(1)= 1:REM STARTPOZICIO
130 FOR I=1 TO N
140 READ L(I):REM A HOSSZ OLVASASA
150 S = S + L(I):REM TELJES HOSSZ SZAMITASA
160 P(I+1) = P(I) + L(I):REM A POZICIO SZAMITASA
170 NEXT
180 :
190 :
1000 DATA 5:REM A MEZOK SZAMA
1010 DATA 25,VEZETEKNEV
1020 DATA 20,KERESZTNEV
1030 DATA 4,MEGYE
1040 DATA 25,HELYSEG
1050 DATA 20,UTCA

READY.
```

Ha a program legfontosabb adatait változókként adjuk meg, a módosítások és illesztések gyerekjátékká válnak. Pillanatok alatt átalakíthatjuk például a címek nyilvántartó rendszerét lemezek nyilvántartásává, az inicializáló rutin megfelelő módosításával. A program azonban még így sem eléggé általános, hiszen ugyanazon program még mindig csak egyfajta adatbázist tud kezelni. Minden feladathoz változtatni kell a programon.
Ez is elkerülhető:
A paramétereket nem a programba, hanem egy file-ba helyezzük el. Így egyetlen program bármilyen szerkezetű adathalmaz kezelésére alkalmas. Futtatás során bekéri klaviatúráról a paraméterfile nevét, és a kiválasztott adatbázis paramétereit beolvassa lemezről. Nézzünk erre is egy példát:

```
P24

100 OPEN 2,8,2,"0:PARAMETER,S,R"
110 INPUT#2,N:REM ADATMEZOK SZAMA
120 DIM P(N+1),L(N):REM POZICIO, HOSSZ
130 P(1)=1:REM STARTPOZICIO
140 FOR I=1 TO N
150 INPUT#2,L(I):REM A HOSSZ OLVASASA
160 S = S + L(I):REM TELJES HOSSZ SZAMITASA
170 P(I+1) = P(I) + L(I):REM POZICIO KISZAMITASA
180 NEXT
190 CLOSE 2

READY.
```

Ezzel az eljárással könnyen készíthetünk szabványprogramokat is. Az adatbázis specifikus adatait csak egyszer kell összeszedni és a paraméterfile-ban tárolni. A program minden indítás után elolvassa a paraméterfile-t és a megfelelő változókhoz hozzárendeli a megfelelő értékeket, illetve szövegeket. Így az XY szabvány pénzügyi-könyvviteli programcsomagból egykettőre megszületik a MAF és FIA cég egyéni pénzügyi könyvelése a cég elvárásainak megfelelő számlakerettel és egyéni figyelmeztető szövegekkel.
A paraméterezésnél azonban még valamit figyelembe kell venni: Mivel a program megírásakor még nem tudjuk, hogy a későbbiekben milyen értékeket kapnak a paraméterek, ezért visszaolvasáskor meg kell vizsgálni a file-ból olvasott értékek érvényességét. Ellenőrzés közben a határesetekre is ügyelni kell, például ha a paraméterek egy ciklust vezérelnek.

Annak ellenére, hogy a végérték kisebb, mint a kezdő érték, a ciklus egyszer lefut, ami esetleg nem szerepelt az eredeti tervekben.

Mi mindent lehet paraméterezni?
Mindig a programozónak kell arról döntenie, hogy milyen messzire kíván menni a paraméterezéssel. Egy azonban biztos: célszerű a lehető legtöbb változót paraméterezni.
Ha például a címek nyilvántartási rendszerébe a mezők számát, megnevezéseit, hosszát paraméterként kezeljük, akkor máris egy teljesen univerzális file-kezelő rendszert készítettünk. Nagyon hasznos, ha a külső egységek típusát is paraméternek tekintjük. Így ugyanis felkészíthetjük a programot arra, hogy különböző nyomtatótípusokkal képes legyen együtt dolgozni. Feltétlenül paraméterezzük az egységszámot, a másodlagos címet és esetleg a file-neveket. Ha a logikai file-szám is paraméter, a nyomtatót automatikus sorhúzással és anélkül egyaránt használhatjuk. Ha ugyanis a logikai file-szám nagyobb, mint 127, akkor a nyomtató minden egyes sor kiírása után új sort kezd [kocsivissza CHR$(13)]. 128-nál kisebb logikai file-számok esetén a sorhúzás elmarad. Igaz, hogy a paraméterezett program elkészítéséhez nagyobb ráfordítás szükséges, ez azonban később egészen biztosan megtérül. Még mindig sokkal jobban járunk, mintha később a program minden részletét át kellene vizsgálnunk ahhoz, hogy esetleg a név mezőjét egyetlen karakterrel megnöveljük.

3.2 Menüvezérlés és overlay-technika

Jól bevált programozási technika a menütechnika, amelynek az a lényege, hogy a feldolgozást egy úgynevezett menü- vagy kiválasztó programmal vezéreljük. Az egyes részproblémákhoz készített programokat a kiválasztó program tölti be. A programoknak az egymást követő betöltését vagy láncolását overlay-technikának nevezzük.
Nézzünk egy példát a menüprogramra:

```
P25

100 REM MENU
110 READ N:REM A PROGRAMOK SZAMA
120 FORI=1 TO N
130 READ A$(I):PRINT I,A$(I):REM NEV ES SZAM
140 PRINT:NEXT
150 PRINT"KEREM VALASSZON!"
160 GET T$: T=VAL(T$): IF T<1 OR T>N THEN160
170 LOAD A$(T),8:REM PROGRAM TOLTESE
180 DATA 3
190 DATA ADATBEVITEL,KIERTEKELES,NYOMTATAS

READY.
```

A menüprogram először beolvassa a menühöz tartozó programok számát és megnevezését. Ha a gépkezelő leüt egy számot, akkor a megfelelő program betöltődik és automatikusan elindul. A program végrehajtása után ismét a menüprogram jelentkezik a képernyőn. Röviden ez a menütechnika lényege. Van azonban két fontos problémá, amire feltétlenül ki kell térni: Az egyik a programok közötti változócsere, a másik pedig a hívó és hívott program mérete. A tárban egymást követő programok csak akkor használhatják egymás változóit, ha a hívó program legalább akkora méretű vagy nagyobb, mint a behívott program. Ha egy programból betöltünk (LOAD) egy másikat, akkor a tárban a programvég mutatói nem változnak és a végrehajtás az új program elejétől indul. Az alábbi program eleget tesz ennek a követelménynek:

```
P26/1

100 REM P26/1
110 REM EZ A PROGRAM NAGYOBB MINT A MASIK
120 A=1000
130 LOAD"P26/2",8

READY.



P26/2

100 REM P26/2
110 PRINT A

READY.
```

Ha azonban az utántöltött program nagyobb, mint az eredeti, akkor a változók egy része felülíródik és helyettük kiszámíthatatlan értékeket kapnánk. Ugyanakkor a változók értékkijelölése során megsérülne a programnak az a része, ami túlnyúlik az első program hosszánál. A változók átadásának további fontos sajátossága: az olyan füzérek átadása, amelyek az

első programban idézőjelek között szerepelnek, tárolási módjuk miatt nem megoldható. A füzérkonstansok meghatározása ugyanis egy mutató alapján történik, amely a programban a szöveg elejére mutat. Ha egy füzérváltozót például a következő utasítással definiálunk az első programban:

```
100 A$="SZOVEG"
```

akkor a változómutató a programszövegre mutat. A következő program utántöltésekor ez a mutató nem változik. Az eredeti helyen azonban most az új programszöveg áll, így a változó tartalma határozatlan. Ismerve az interpreternek ezt a sajátosságát, a problémát könnyen megoldhatjuk. Csupán annyit kell tennünk, hogy a programból átmásoljuk a szöveget a felső RAM tartományba, ahol az operációs rendszer a szövegváltozókat tárolja.
Az áthelyezésre alkalmas utasítás:

```
100 A$="SZOVEG" + ""
```

Az üres füzérrel végzett művelet az A$ eredeti tartalmát nem módosítja, de a változót áthelyezi. Hasonló meggondolások érvényesek a függvények definícióira is, a mutató itt is a programban lévő DEF utasításra mutat. Ebben az esetben az egyetlen megoldás az, hogy a második programban a függvényt mégegyszer definiáljuk:

```
100 DEF FN A(X)= 0.5 * EXP (-X*X)
```

Az egymást követő programok közötti paramétercserét tehát csak akkor tudjuk minden különösebb előkészítés nélkül megoldani, ha a sorrendben második (utántöltött) program mérete nagyobb az első program méreténél. Ha a második program nem akarja használni az első program eredményeit, akkor egyetlen BASIC sorral megoldható az utántöltés abban az esetben is, ha az első program nagyobb a másodiknál:

```
POKE 45, PEEK(174): POKE 46, PEEK(175): CLR
```

A második program első soraként a fenti BASIC sort kell beilleszteni. A CLR utasításra feltétlenül szükség van, ebből azonban az következik, hogy a tárban a változók tartalma törlődik.
Ezzel a módszerrel ugyan a változókat elveszítjük, de tetszőleges programokat betölthetünk egymás után. A paramétercsere, ha mindenképpen szükséges, megoldható egy lemezen tárolt paraméterfile segítségével is.
A programok utántöltésének másik, kevésbé elegáns módja a közvetlen betöltés szimulálása. A betöltő utasítást beírjuk a billentyűzet pufferbe úgy, hogy először PRINT utasítással kiírjuk a képernyőre egymás alá a LOAD és a RUN parancsokat, a billentyűzet pufferébe pedig POKE utasítással betöltünk egy HOME és két RETURN karaktert. A fenti BASIC sorokat a betöltő programban END utasítással kell zárni.
A végrehajtás során az operációs rendszer parancs üzemmódban beolvassa a billentyűzet puffer tartalmát, a LOAD és a RUN parancsokat, majd a RETURN hatására betölti és elindítja a programot. A változók tartalma automatikusan törlődik, hiszen a végrehajtás parancs üzemmódban zajlik.
A megoldás azért nem nevezhető kifejezetten elegánsnak, mert a LOAD és a RUN parancs fizikailag is megjelenik a képernyőn, és elrontja a képmaszkot.
Feltehetően az Olvasó ennek ellenére szeretné látni a megoldást:

P27

```
1000 PRINT CHR$(147)"LOAD"CHR$(34)"PROGRAM"CHR$(34)",8"
1010 PRINT:PRINT:PRINT:PRINT
1020 PRINT"RUN"
1030 POKE 631,19:POKE 632,13:POKE633,13
1040 POKE 634,19:POKE 635,13:POKE636,13
1050 POKE 198,6:END

READY.
```

A kétféle megoldást összehasonlítva látjuk, hogy az első jóval egyszerűbb, mint a második. Az első módszer szerint a betöltő programba az

```
1000 LOAD"2.PROGRAM",8
```

utasítást, a betöltött programba pedig a fenti két POKE és a CLR utasítást kell beírni.

3.3 A funkcióbillentyűk szerepe a fejlett programtechnikában

Az alapgép jobb oldalán szembetűnő és könnyen elérhető helyen találhatók a funkcióbillentyűk 1-től 8-ig számozva.
Mire lehet ezeket a billentyűket használni, és miért éppen funkcióbillentyűknek nevezzük őket?
Ezekre a kérdésekre adunk választ ebben a fejezetben.
A funkcióbillentyűk legelőnyösebb tulajdonsága furcsa módon az, hogy semmilyen jelentésük nincs. A programozás oldaláról nézve ez azt jelenti, hogy tetszőleges feladattal ruházhatjuk fel őket, tehát szabadon programozhatóak. Néhány alapszabályt azonban a programozás során szem előtt kell tartani.
Egyfelől tényleg nagyszerű dolog az, hogy bármilyen funkciót hozzárendelhetünk ezekhez a billentyűkhöz. Azonban tekintettel kell lennünk a program használójára is, akinek ezeket a funkciókat fejben kell tartania.
Ezért egyrészt körültekintően meg kell gondolni, hogy melyik billentyűt milyen feladattal ruházzunk fel, másrészt arra kell törekednünk, hogy egy programon belül, de még jobb, ha az összes programban azonos legyen az egyes billentyűk feladata.
Vizsgáljuk meg ebből a szempontból például a cégünk által gyártott TEXTOMAT és DATAMAT rendszereket. Mindkét rendszer előszeretettel használja a funkcióbillentyűket. Két billentyűnek mindig, azaz minden programrészben és minden időpontban azonos a jelentése. Az F1 billentyű mindig azt jelzi, hogy egy bevitel befejeződött vagy semmisnek nyilváníttatott. Az F2 billentyű jelentése pedig mindig az, hogy egy programrészből kilépünk.
Rendkívül előnyös, ha a gépkezelőnek mindig ugyanazt a billentyűt kell megnyomnia ugyanazon műveleteknél, mert így átállási nehézségek nélkül válthat át az egyik programról a másikra. Milyen könnyítést jelentenek a funkcióbillentyűk a programozás során?
A funkcióbillentyűket a programban bármikor lekérdezhetjük anélkül, hogy a feldolgozást ezzel a legkisebb mértékben is zavarnánk. Ugyanezt egyetlen más billentyűről sem mondhatjuk el, hiszen a gépkezelő a többi billentyűt munka közben használja.
A funkcióbillentyűk használatát ergonómiai szempontok is alátámasztják. Elhelyezkedésük a billentyűzeten szinte kizárja, hogy a gépkezelő hibát vétsen. A belső kódrendszer is támogatja a funkcióbillentyűk programozását azzal, hogy ASCII kódjai a számsorban egymás után következnek:

F1 – 133
F3 – 134
F5 – 135
F7 – 136
F2 – 137
F4 – 138
F6 – 139
F8 – 140

A következő programrészlet bemutatja a lekérdezést:

```
10 GET E$ : IF E$ = CHR$(133) THEN 1000 : REM F1
20 IF E$ = CHR$(137) THEN 9999 : REM F2
30 GOTO 10
 :
```

```
1000 REM A PROGRAM ITT FOLYTATODIK
   :
9999 END : REM A PROGRAM VEGE
```

A 10-es sorban megvizsgáljuk, hogy a gépkezelő az F1 billentyűt ütötte-e le. Ha igen, akkor az E$ változóban CHR$(137) karakter található, tehát a programot az E$ tartalma alapján elágaztatjuk. A 20-as sorban pontosan ugyanezt csináljuk az F2 billentyűre vonatkoztatva. A funkcióbillentyűk lekérdezése a programból semmivel sem nehezebb, mint bármelyik más billentyűé.

3.4 Teszteljük programjainkat! (A program használói nem kísérleti nyulak!)

Melyek a programozó életének legsötétebb pillanatai?
Minden kétséget kizáróan azok, amikor csörög a telefon és programjának üzemeltetője újabb és újabb hibajelzésekről számol be, esetleg ügyvédjével fenyegetőzik.
Az ilyen kellemetlen eseményeket egyetlen módon előzhetjük meg, ha a programokat átadás előtt alapos tesztelésnek vetjük alá. Legjobb, ha a végső tesztelést a program jövendőbeli használóira bízzuk.
Közbenső tesztelési lépéseket persze már a program készítésének folyamatába is be kell iktatnunk.
Minden programrészt vagy szubrutint érdemes még összefűzés előtt önállóan kipróbálni, ezzel feltétlenül megkönnyítjük a későbbi hibakeresést. A strukturált programozási technika előnyei a tesztelés során feltétlenül érvényesülnek.
Sajnos a C 64-es utasításkészletében nincsenek a teszteléshez megfelelő segédeszközök, ún. TOOLKIT utasítások, mint pl. a TRACE, a FIND vagy a DUMP utasítás. Pedig igazán jó programokat ezek nélkül nagyon nehéz készíteni. Ha már bonyolultabb, komolyabb programokat írunk, mindenképpen érdemes egy olyan segédszoftvert beszerezni, amely tartalmazza ezeket az utasításokat. Különösen hasznos közülük a tesztelési fázisban a TRACE utasítás, amellyel végrehajtás közben minden programlépést kiírathatunk a képernyőre. Ugyancsak megnehezítheti a hibakeresést a dokumentáció hiánya. A dokumentációt legjobb a programírással párhuzamosan elkészíteni. Ha a program befejeztével kezdjük a programot dokumentálni, lehet, hogy az eredmény hiányos lesz, időközben ugyanis néhány dologról megfeledkezünk. Gondoljunk pl. a változónevek többszöri felhasználásából eredő hibákra. Ilyesmi elő sem fordulhat, ha a programozás során végig ott áll előttünk a változólista.
A file-leírásokat, képernyőmaszkokat pedig tulajdonképpen a program begépelése előtt kell elkészíteni.
Nagyon sok programozó számára a dokumentáció készítés felesleges teher, és ha meg is írja, többnyire a program befejezése után teszi és inkább rosszul, mint jól.
Vannak olyan alattomos hibák is, amelyek eredetére a tesztelés során sem derül fény, ekkor az egyetlen megoldást egy-egy programrész teljes újraírása jelenti.
Sajnos még mindig előfordulhat, hogy egy-egy megmagyarázhatatlan hiba a Commodore 64-es operációs rendszerében vagy külső egységeiben rejlő fogyatékosságra vezethető vissza. Ilyenkor nem marad más hátra, mint a megfelelő programrész teljesen új koncepció alapján történő újraírása.
Talán tanulságos lesz az Olvasó számára a programtesztelés alábbi epizódja: a könyvünkben bemutatott QUISAM file hozzáférési eljárást, készítője Klaus Gerits annak rendje s módja szerint letesztelte és hibátlannak találta. Már majdnem be is került változatlan formában a könyvbe, amikor a következő történt. A könyv utolsó részében található használói programnak kellett illusztrálnia a QUISAM alkalmazását. Így Wolfgang Schellenberger a QUISAM alapján írt egy irodalomkezelő programot. És mi történt – programja nem működött, mert a QUISAM-ban volt még két apró hiba! Ezek a hibák az utolsó pillanatig rejtve maradtak, mert csak egészen sajátos összefüggésben volt szükség a hibás részekre. Saját tapasztalatainkra hagyatkozva is azt kell ajánlanunk az Olvasónak, hogy programjait ne csak saját maga tesztelje részletesen és behatóan, hanem egy többé-kevésbé elfogulatlan harmadik személynek is adja át tesztelésre. A programozásnál is létezik az „üzemi rövidlátás", ami meggátolhat még a komolyabb hibák felfedezésében is.
Óvatosan kell eljárni a kész programok utólagos módosításánál és javításánál is. Nem ritka

eset, hogy egy érett, hibátlan program készítőjének új zseniális ötlete bosszantó hibaforrássá válik. A program módosítása előtt a dokumentáció és a változólista alapján gondosan meg kell vizsgálni, hogy a módosítás mely programrészekre gyakorolhat hatást. Természetesen minden módosítást ismét megfelelően dokumentálni kell. Csak így kerülhetjük el a kellemetlen meglepetéseket. A mások által használt program állandó módosításaitól amúgy is óva intünk mindenkit. Sokkal ésszerűbb, ha nagyobb időközönként kiadjuk a program új változatát, ami tesztelt formában az összes módosítást tartalmazza. A program üzemeltetői is értékelni fogják, ha nem naponta kell a program újdonságaihoz igazodniuk.

3.5 A kezelési utasítás a programozó névjegye

Sok programozó ugyan egészen kitűnő programokat ír, de sajnos elfeledkezik arról, hogy a programokkal más, halandó embernek is dolgoznia kell. Az, hogy ez csak jó és részletes dokumentáció és kezelési utasítás alapján lehetséges, arról úgy tűnik, nagyon kevesen hallottak.
Azt is figyelembe kell venni, hogy egy program vevője az esetek 99%-ában először a kezelési utasítást olvassa át és így nyeri első benyomását a programról. Nagyon gyakran előfordul, hogy az értékes programokat nem mutatják be az üzletben, így a vevő egyedül és kizárólag a kezelési utasítástól teszi függővé, vásárol-e vagy sem. Kereskedelmi szempontból egyáltalán nem elhanyagolható szempont!
Ha valakit az üzleti szempontok nem érdekelnek, gondoljon arra, hogy a jó kezelési utasítás mennyi időt és energiát takarít meg programozónak és használónak egyaránt.
Az Olvasót bizonyára meggyőztük arról, hogy kezelési útmutató nélkül nem lehet a programot útjára bocsátani. Nincs más hátra, mint tisztázni a jó kezelési útmutató ismérveit.
A legfontosabb, hogy az útmutató kezdő gépkezelők számára készüljön. Mindig abból a feltételezésből kell kiindulni, hogy a gép előtt egy programozásban teljesen járatlan ember ül.
Kezdjük az útmutatót azzal, hogy a szövegben előforduló szakkifejezések értelmét megmagyarázzuk. Persze az a legjobb, ha eleve kerüljük a szakkifejezéseket. Gondoljunk arra, hogy ami a szakember számára magától értetődik, a kezdő számára feltehetőleg teljesen ismeretlen.
A kezelési útmutató áttekinthetőségét szolgálja a jó tartalomjegyzék, a rendezett szójegyzék, és a világos, tagolt mondatszerkesztés. A kézikönyvet úgy kell elkészíteni, hogy abból a használó minden munka közben felmerülő kérdésére gyorsan, különösebb fáradság nélkül választ kapjon.
Az útmutató készítése közben számoljunk az olvasó elfáradásával. Ennek ellensúlyozására a sok egyszerű példán kívül a grafikák és képernyőre íratások sem elhanyagolhatóak. Az anyag sokkal könnyebben emészthető, ha feltűnő grafikákkal tarkítjuk. A képek legyenek tiszták, pontosak és ösztönző erejűek. Az Olvasó számára a grafikus ábrák apró sikerélményeket jelentenek a megértésben és oldják a fáradtságérzetet.
Az alapvető követelmények tisztázása után próbáljuk meg elképzelni az útmutató szerkezeti felépítését.
Engedjék meg, hogy tegyünk erre egy javaslatot, és azt egy példán keresztül bemutassuk, természetesen a teljesség igénye nélkül. A vázlat az általunk alkalmazott, a gyakorlatban jól bevált módszereket tükrözi.
A kézikönyv durván két részből áll. Az első rész az ún. gyakorlati kézikönyv tartalmazza a gépkezelő által elvégzendő teendőket. Ebben a részben lépésről lépésre ismertetjük az olvasóval (a kézikönyv olvasójával) a program üzemeltetésével kapcsolatos feladatokat. Ismertetjük a legfontosabb szakkifejezéseket, majd részletesen leírjuk, hogyan kell a programot indítani (kezdve a számítógép bekapcsolásával). Minden szükséges lépésre ki kell térni, beleértve a billentyűk leütésének sorrendjét is. Például:
Helyezze be a lemezt az egységbe, zárja le az egység ajtaját és gépelje be a következőket:

```
LOAD "*",8
```

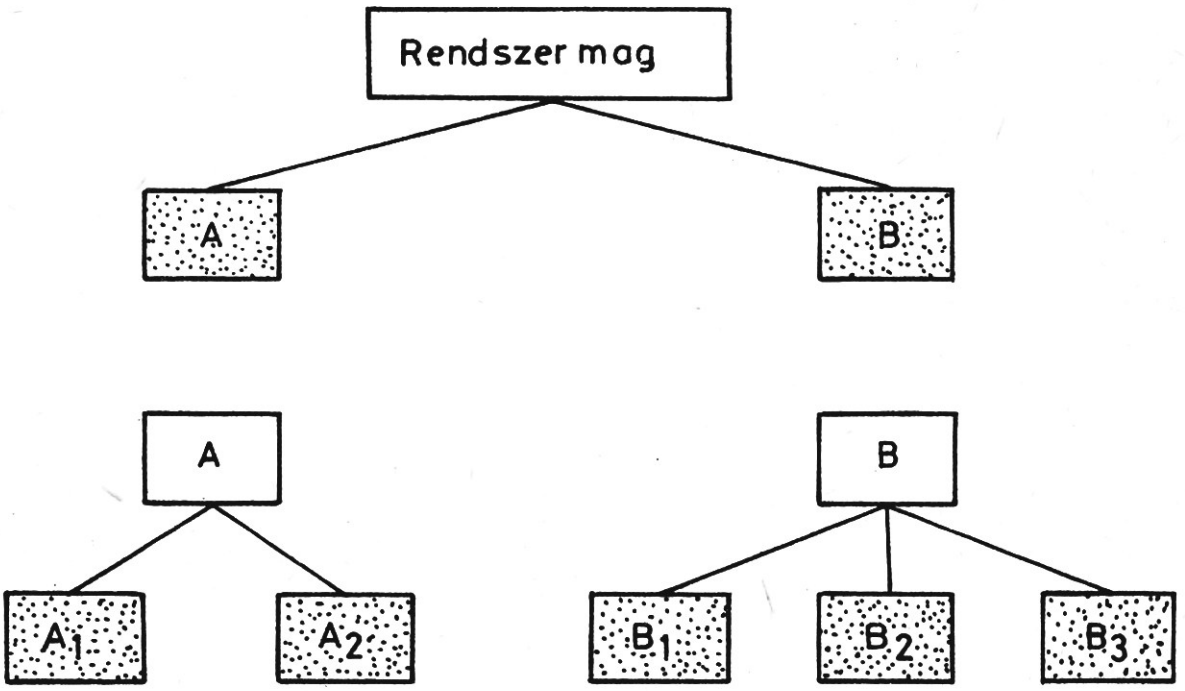
Nyomja le a RETURN billentyűt.
Néhány másodperc elteltével a villogó kurzor ismét megjelenik a képernyőn. Most gépelje be a következőt:

Nyomja meg ismét a RETURN billentyűt.

Lehet, hogy a szöveg stílusát sokan túlságosan elemi szintűnek tartják, de nem feledkezhetünk meg arról, hogy a számítógép előtt adott esetben olyan valaki ül, aki most dolgozik először az „ördöngős masinával". A program indításának leírása után adhatunk néhány általános útmutatót a program kezeléséhez. Pl. hogyan dolgozik a program, mire kell különösen ügyelni.
A bevezetést követheti az egyes tevékenységek részletes leírása, példákkal gazdagon illusztrálva.
A szöveg megfogalmazása közben ne arra törekedjünk, hogy a használót a program működésének apró finomságairól tájékoztassuk, hanem arra, hogy a lényeges teendők semmiképpen ne sikkadjanak el. Két dologra kell feltétlenül ügyelni: a szöveg legyen tökéletesen közérthető, és mindenütt hívja fel a figyelmet a hibalehetőségekre, és a javítás módjára. Ugyanakkor az egész ismeretanyag kis, könnyen emészthető adagokra legyen osztva, úgy, hogy az olvasó lépten-nyomon sikerélményekhez jusson. Nagyon jó, ha a könyvben pontosan ábrázoljuk a képernyő tartalmát. Ha a használó azonosítja a könyvben szereplő ábrát a saját képernyőjén látottakkal, megbizonyosodik arról, hogy mindent jól csinált, és folytatja a munkát. A kezelési útmutató minőségét aszerint lehet megítélni, hogy mennyi ideig képes valaki vele folyamatosan dolgozni.
Az útmutató második része az ún. felhasználói kézikönyv. Ez a rész témák szerint csoportosítva tartalmazza az egyes programrészek leírását.
Minden leírást azzal kell kezdeni, hogy milyen eljárással juthatunk a kiválasztott programrészbe. Ezután írjuk le, hogy mire szolgál a programrész, milyen hibát lehet véteni, és végül hogyan léphetünk ki ebből az egységből. Közben semmit sem szabad elnagyolni vagy kihagyni, a használót minden eshetőségre fel kell készíteni.
Az útmutatónak ez a része lehet terjedelmes, persze a helyes arányokat mindig meg kell találni; ha túlságosan hosszan magyarázunk, a lényeg elvész és az olvasó eltéved a részletekben. Minden útmutatóhoz csatolni kell egy pontos hibajegyzéket. Ebben fel kell sorolni az összes lehetséges hibajelenséget, a hiba okát és a javítás módját.
Végül nem maradhat el az áttekinthető tartalomjegyzék és – ami nem ennyire kézenfekvő – egy ábécébe rendezett tárgymutató. Az utóbbinak tartalmaznia kell minden fontos fogalmat, programnevet, utasítást stb., ellátva azzal az oldalszámmal, ahol a megfelelő magyarázat található.
Nem haszontalan az utasítások összefoglalása, a hozzájuk fűzött pár szavas magyarázattal. Kisebb programok leírásánál ezt elhagyhatjuk.
Még egyszer összefoglalva: az útmutató legfontosabb tulajdonságai: közérthetőség, világos szerkezet, esztétikus külső.
Aki mindezek után kétségbe vonja, hogy a fent ismertetett alaposságú kezelési útmutatóra valóban elengedhetetlenül szükség van, az gondoljon arra, hogy nagyon gyakran a program sorsa múlik az útmutató minőségén.

3.6 Felülről lefelé vagy lentről felfelé? (TOP-DOWN vagy BOTTOM-UP?)

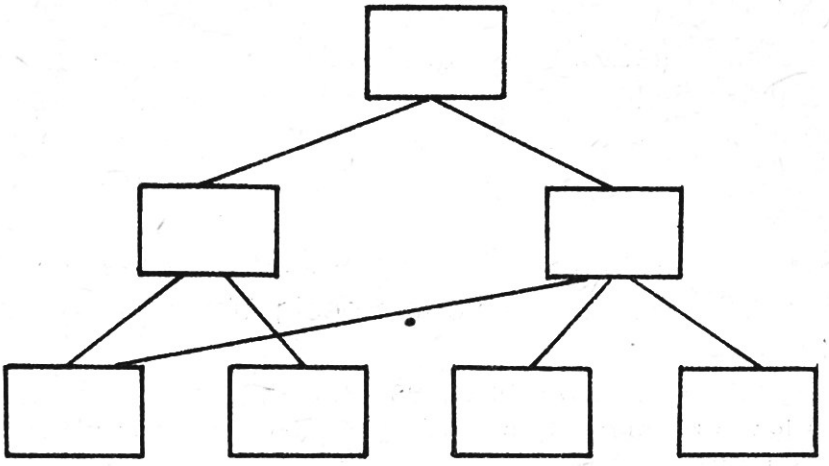
A programozás elméleti kérdéseiből már csak alig néhány maradt tisztázatlanul. Az egyik ezek közül a programrészek megírásának sorrendje. Két ésszerű módszer közül választhatunk, az egyik a TOP-DOWN (felülről lefelé), a másik a BOTTOM-UP (alulról felfelé) építkezés módszere.
A TOP-DOWN módszer szerint, elindulva a program külső szerkezetéből folyamatosan haladunk a részletek kidolgozása felé. Egy adatfeldolgozási feladat esetében ez azt jelenti, hogy először megírjuk a főmenüt, a szubrutinok helyére pedig átmenetileg egyetlen REM sort és egy RETURN utasítást írunk.
Ezt követően sorra vesszük a szubrutinokat, és átgondoljuk, hogy ezekhez még milyen további alprogramokra lesz szükségünk, a kidolgozást pedig ezekkel folytatjuk. A módszert a következő ábra szemlélteti:



A TOP-DOWN módszerhez nagyon jól használhatjuk az előzetesen elkészített folyamatábrát vagy struktogramot.

A BOTTOM-UP programozás a program legalsó szintjéről indul. Ezt a módszert követve a programozó először a leggyakrabban használt programrészeket készíti el, mint amilyen pl. egy rekord beolvasása, vagy összeállítása. Az elkészült programrészeket a következő szinten levő programok, a következő szint programjait pedig egy magasabb szint programjai hívhatják. Ennél a módszernél a főprogram is csak egy a programegységek sorában.

A módszer diagramja:



A programozói gyakorlatban előfordul a két módszer egyfajta keverése is. Elkészítve a program vázát, elkezdjük kidolgozni a szubrutinokat, és ezeket egyenként azonnal teszteljük is. Így van reményünk arra, hogy a végső fázisban a program már hibátlanul fog működni. Persze kisebb programok megírását nem kell feltétlenül elméletileg megalapozott módszerekkel támogatni. Kezdve a program legfontosabb egységével, folyamatosan haladhatunk a részletek kidolgozása felé.

3.7 A logikai típusú változók – nyolc lehetőség byte-onként

Az adatfeldolgozó programokban gyakran előfordul, hogy a továbblépéshez a gépkezelőnek egy eldöntendő kérdésre kell válaszolnia. Például Nő/Férfi, Nőtlen/Házas ... Igen/Nem stb. A beviteli mező – az adatlap egyetlen rubrikája – mindössze két különböző értéket vehet fel. A két lehetséges értékű változókat logikai- vagy döntési változóknak nevezzük.
Természetesen óriási helypocsékolás a programban az ilyen változó számára egy teljes byte-ot elfoglalni.
Minden byte nyolc bitet tartalmaz, melyek mindegyike két értéket, 0-t vagy 1-et vehet fel. Egy byte-ban tehát összesen nyolc döntési változót tárolhatunk.
A megoldás gazdaságossága nem kétséges, de hogyan lehet a feladatot programozástechnikailag kezelni?
Tegyünk egy kis kitérőt a matematikába, a kétértékű algebra, vagy másként Boole algebra területére. Arra nincs szükségünk, hogy az elmélet mélységeit bolygassuk, számunkra a Boole algebrának a gyakorlati használhatósága érdekes.
Két művelettel kell megismerkednünk, a logikai AND (ÉS), ill. OR (VAGY) műveletekkel.
A számolások közben gyakran szükségünk lesz a byte egyes bitjeinek kettes számrendszerbeli helyiértékére:

```
128 64 32 16 8 4 2 1
```

A fenti számok 2 hatványai 0-tól 7-ig.
Az OR művelet segítségével egy byte bármely bitjének értékét 1-re állíthatjuk. Tegyük fel, hogy a B változót (1 byte) felosztjuk nyolc döntési változóra. Ha azt akarjuk, hogy pl. az első döntési változó értéke 1 legyen, de úgy, hogy közben a többi ne változzon, a következő műveletet kell végrehajtanunk:

```
B=B OR 1
```

A művelet eredményeként az első bit értéke 1 lesz, függetlenül attól, hogy előtte 0, vagy 1 volt. Tegyük fel, hogy B eredeti tartalma 254 volt decimálisan, melynek bitmintája a következő:

```
11111110
```

A számítás eredménye:

```
11111111
```

A B értéke az OR művelet után 255.
Ha a B változó valamely bitjét 1-re szeretnénk változtatni, nem kell mást tenni, mint a megfelelő bit helyiértékéhez tartozó számérték és az eredeti változó között OR műveletet végezni. Nézzünk erre még egy példát: Ha B tartalma 163, azaz

```
11000011
```

és az 5. bitre szeretnénk 1-et kapni, mivel az 5. bithez kettő negyedik hatványa, azaz 16 tartozik, a megfelelő művelet:

```
B=B OR 16
```

Az eredmény 163+16=179, tehát B értéke 179 lesz.
Hasonlóan egyszerű a bitek értékének törlése (0-ra állítása) is. A megoldás menete: kivonjuk 255-ből 2-nek a kiválasztott helyiértékhez tartozó hatványát és az így kapott szám, ill. az eredeti változó között AND műveletet végzünk. Például: ha a B változó tartalma 161, és a 6. bitet akarjuk törölni, a következőképpen kell eljárni: a B változó bitmintája:

10100001

Kivonjuk 255-ből a 6. bitnek megfelelő értéket:

255−32=223

Végül elvégezzük az AND műveletet:

B=B AND 223

Az eredmény:

10000001

B decimális értéke tehát 129.

A fentiek igazolják, hogy a döntési változók kezelése rendkívül egyszerű, kis befektetéssel sok helyet takaríthatunk meg. Még egy programozástechnikai ötlet: a B változó értéke csak 0-tól 255-ig terjedhet, így fölösleges numerikus változóban tárolni. A CHR$ utasítás lehetővé teszi az egy byte-os tárolást.

3.8 A dátum, mint különleges adattípus

Bármilyen célra készítünk programot, a dátum majdnem minden esetben szerepel az adatok között, érdemes tehát célszerű programozásával külön foglalkozni.
A dátum programozási kérdéseihez tartozik a tárolás, a rendezés, a hét adott napjaihoz tartozó dátum kiszámítása stb.
Tegyük fel, hogy a dátumot három változóban tároljuk: az N változó a napot, a H változó a hónapot, az E változó pedig az évet tartalmazza évszázad nélkül. A legegyszerűbb az lenne, ha ezeket a változókat külön-külön tárolnánk. Figyelembe véve a numerikus változók helyigényét, érdemes egy takarékosabb tárolási mód után kutatni. A változók értéke rendre 1-től 31-ig, 1-től 12-ig és 0-tól 99-ig terjedhet. Mivel a számok egyike sem lehet 255-nél nagyobb, a CHR$ függvénnyel mindegyiküket átalakíthatjuk karakteres formába. A dátumhoz rendelhető karakterláncot a következő BASIC sorral rakhatjuk össze:

```
1000 DA$=CHR$ (E)+CHR$ (H)+CHR$ (N)
```

A DA$ változó tárolásához mindössze három byte-ra van szükség. A visszaalakítás is nagyon egyszerű:

```
2000 E = VAL (LEFT$(DA$,1))
2010 H = VAL (MID$ (DA$,2,1))
2020 N = VAL (RIGHT$ (DA$,1))
```

A dátum karakteres tárolása a rendezés szempontjából is előnyös. Pontosan ugyanúgy kezelhetjük a dátumot tartalmazó változót a rendezés során, mint bármelyik füzért (például vezetéknevet). A 2. fejezetben ismertetett rendezési eljáráskor pl. egyszerűen betöltjük a rendezendő dátumokat egy karakteres tömbbe. Gyakori feladat az, hogy tetszőleges dátum alapján meg kell határozni, hogy az a hét melyik napjára esett. A következő program (P.28.) meghatározza a napot a Gergely naptárat véve alapul. A programot szubrutinként is használhatjuk. A dátumot a TA (nap), MO (hónap) ill. JA (év) változókban kell a szubrutinnak átadni, az eredményt pedig a T$ változóban kapjuk vissza. Lekérdezhetjük a nap sorszámát is (0-tól 6-ig, azaz szombattól péntekig), amelyet a szubrutin az IN változóba helyez el.

```
P28

100 REM NAPOK KISZAMITASA DATUM ALAPJAN
110 INDEX=NAP
120 INDEX=INDEX*31(HO-1)
130 INDEX=INDEX+365*EV
140 IF HO>2 THEN INDEX=INDEX-INT(0.4*HO+2.3):GOTO 160
150 EV=EV-1
160 INDEX=INDEX+INT(EV/4)
170 X=INT(EV/100)+1
180 INDEX=INDEX-INT(0.75*X)
190 INDEX=INDEX-INT(INDEX/7)*7
200 RESTORE
210 FOR J=0 TO INDEX
220 READ T$
230 NEXT J
240 DATA SZOMBAT,VASARNAP
250 DATA HETFO,KEDD
260 DATA SZERDA,CSUTORTOK
270 DATA PENTEK

READY.
```

3.9 Az adatok tömörített tárolása – egy rekordban több mint 255 karakter!

A Commodore 64-esen mint tudjuk, maximum 255 karaktert tartalmazó adatblokkokkal dolgozhatunk.
Ennek oka a nyolcbites társzervezés.
Ha egy adatblokkban több mint 255 karaktert akarunk tárolni, akkor ezt a fizikai határt valahogyan át kell lépnünk.
Két lehetőség között választhatunk:

1.) A fizikai határt logikailag lépjük át. Ez azt jelenti, hogy két vagy több fizikai adatblokkot egy logikai egységként kezelünk. Ez a módszer szöveges típusú változók tárolása során célszerű.

2.) A másik módszer az adatok tömörítése. Ez azt jelenti, hogy egy byte-ban több információ-egységet tárolunk. Ezzel a módszerrel vagy a számok ábrázolási módját változtatjuk meg, vagy pedig kihasználjuk, hogy az adatok mindig egy megadott számtartományba esnek.

A második módszer alkalmazására már bemutattunk két példát: az egyik a dátum tömörített (3 byte-on történő), a másik a logikai változók (1 byte-on történő) tárolása volt.
A tömörített tárolásnak az a módja, amelyet most ismertetünk, teljesen független attól, hogy az adatok mekkora számtartományba esnek.
Egy decimális számjegy bináris ábrázolására mindig elegendő 4 bit:

```
0000  0
0001  1
0010  2
0011  3
0100  4
0101  5
0110  6
0111  7
1000  8
1001  9
```

A négy bit lehetséges kombinációi közül még maradt 6, amelyeket felhasználhatunk az előjelek és a tizedespontok jelölésére, pl.:

```
1010 = "-"
1011 = "."
```

Az elmondottak alapján egy byte két előjeles decimális számjegy tárolására elegendő.
A következő program beolvas egy kétjegyű számot, és meghatározza a hozzárendelt tömörített alakot, vagyis az átalakítás után kapott byte tartalmának megfelelő karakter kódját.

P 28/1

```
100 Z$=""
110 FOR I=1 TO 2
120 GET E$ : IF E$ "O" ORE$ "9" THEN 120
130 2$=2$+E$ : NEXT I
140 B=(ASC(LEFT$(Z$,1))–48)*16
150 B=B+ASC(RIGHT$(Z$,1))–48
160 B$=CHR$(B) : PRINT B
```

A 110-től a 130-ig terjedő sorok a számjegyeket karakteresen beolvassák, és az eredményt a Z$ változóba rakják.
A 140-es és a 150-es sorokban a számjegyeknek megfelelő ASCII kódokból levonunk 48-at, az eredmény egy 0 és 9 közé eső szám. A magasabb helyiértékű számjegy a felső 4 bitre kerül, a kapott eredményt megszorozzuk 16-tal.
A második számjegy átalakítása azonos az elsővel, a szorzástól eltekintve.
Az eljárás eredményeként kapunk egy 1 byte-os, azaz 0 és 255 közé eső számot, amelyet a CHR$ függvénnyel karakterré alakítunk. Végül is egy kétjegyű decimális számot egy byte-on sikerült ábrázolnunk.
Az eljárás megfordítottja is nagyon egyszerű.
Kiindulunk az egy byte-on tárolt karakter kódjából (B), és előállítjuk belőle az eredeti füzért (Z$):

P 28/2

```
200 Z$=""
210 Z=B AND 240 : Z=Z+48 : Z$=CHR$(Z)
220 Z=B AND 15 : Z=2/16+48 : Z$=Z$+CHR(Z)
230 PRINT Z$
```

A visszaalakítás első lépéseként AND logikai művelettel leválasztjuk az alsó négy bitet, majd elvégezzük az előző program számítási lépéseit fordított sorrendben. Ugyanazt tesszük a felső négy bittel is.
A betűket is ábrázolhatjuk tömörített formában, ennek azonban csak akkor van értelme, ha a jelkészlet egy szűkített részével dolgozunk, pl. csak nagy- vagy csak kisbetűkkel.
A tömörítéssel karakterenként 1-2 bitet takaríthatunk meg, ami önmagában ugyan kevésnek tűnik, de nagy adattömegnél már jelentős méreteket ölthet.

Amint az Olvasó látja, nagyon sokféle módja van a takarékosságnak. Hogy mikor, melyik módszer a legalkalmasabb, azt mindig a feladat egyedi sajátossága szabja meg.
Az ötletek sorát biztosan az Olvasó is gazdagítja majd.

4. FEJEZET
PROGRAMPÉLDÁK

4.1 Bevezetés

Az elméleti eredmények sohasem teljes értékűek, ha gyakorlati alkalmazásokkal nem támasztjuk alá őket.
Ebben a fejezetben az előzőekben tárgyalt témakörök szinte mindegyikéhez csatolunk egy-egy programot, amelyben az Olvasó megtalálja elméleti javaslataink programtechnikai megoldását. A paraméterfile-ok hasznát pl. egy raktárkezelő programrendszer ismertetésével igazoljuk, ami ugyanakkor a relatív file-ok programozási módszereit is tartalmazza. Az irodalomjegyzékeket nyilvántartó program a file-kezelés "új" módszerét, a QUISAM adattárolási struktúra alkalmazását reprezentálja. Mellékelünk egy kis szövegfeldolgozó programot is ami – annak ellenére, hogy nem helyettesíti a szövegfeldolgozó rendszert – közvetlen használatra is alkalmas. A programok mindegyikét úgy terveztük, hogy a szemléltetésen kívül a gyakorlatban is alkalmazhatóak legyenek. Szerkezeti felépítésük egyszerű, áttekinthető és így könnyen módosítható. Olyan univerzális programot természetesen nem tudtunk írni, ami minden Olvasó igényeinek megfelel.
Nem is javasoljuk, hogy mindenki azonnal begépelje a látszólag igénye szerint megírt programot.
Sokkal nagyobb lesz a várható eredmény, ha az Olvasó először alaposan átnézi a programlistát, a megfelelő részeket saját elképzeléseihez igazítja, és csak ezt követően kezdi el a begépelést. Így nemcsak használhatóbb program tulajdonosa lesz, hanem munka közben elmélyítheti az első három fejezet elméleti ismeretanyagát. A programok módosítását egyszerűsíti a mellékelt programleírás. A magyarázatban kitértünk a programok minden lényeges egységének céljára, a megvalósítás módjára.

Útmutatás a programmagyarázatok felépítéséhez.
Természetesen minden programról teljes listát készítettünk, és igyekeztünk elkerülni a grafikus és különleges karakterek használatát, ahol lehetett, ezeket a CHR$ függvénnyel helyettesítettük. Az Olvasó minden program előtt talál két kiegészítést:
A kiegészítés első részében egyrészt bemutatjuk a program alkalmazási területeit, elsősorban azok számára, akik saját céljaikra szeretnék használni a programgyűjteményt. Másrészt ezt a leírást mintának szántuk a kézikönyvek készítéséhez.
Javaslataink nem tekintendők szentírásnak, mi már akkor is elégedettek leszünk, ha az általunk közölt minták alapján szárnyra kap az Olvasó fantáziája.
A második rész a program részletes felépítésével foglalkozik. Minden speciális programozási eljárást pontosan elmagyarázunk. Ez persze nem azt jelenti, hogy minden PRINT és minden INPUT utasítás szükségességét megindokoljuk, de az biztos, hogy minden, a programmal kapcsolatban lehetséges kérdésre választ adunk. Jó munkát és jó szórakozást kívánunk a további fejezetekhez!

4.2 Paraméterezett raktárnyilvántartás

1. RÉSZ:

Napjainkban az elektronikus adatfeldolgozás sokféle adminisztratív tevékenységet lényegesen leegyszerűsít. Egyik ilyen adminisztratív terület a raktárnyilvántartás. A tulajdonos napra készen szeretné tudni, hogy az egyes termékekből, anyagokból mennyi van raktáron. Állandóan figyelnie kell, hogy az értékesítés, illetve szállítás hogyan változtatja meg a raktári készletet. A következő program elvégzi a raktáros helyett a ki- és bevételezések könyvelését. A program egyik sajátossága – amint azt már a neve is mutatja – a paraméterfile használata. A paraméterfile általánosabbá teszi a feladat megoldását.
Képzeljük el, hogy a különböző cégek, különböző értéktöbblet-adókulcsokkal számolnak. Általában az értéktöbblet-adókulcs 7%, ill. 14%. Ha a programokban fix értékekkel dolgozunk, változás esetén minden hivatkozásnál módosítást kell végeznünk. Különösen nagy gondot jelenthet a módosítás, ha a programot nem egyszeri használatra, hanem eladásra készítettük.
Minden vállalat, aki megvásárolja a programot, másképpen fogja kérni a képernyőn megjelenő ill. a nyomtatott listák fejléceit. Ha a szövegeket fixen beépítjük a programba, minden eladás újabb alapos javítással, foltozgatással jár. A hivatásos programozó nem engedheti meg magának ezt az időpocsékolást.
A szoftver cégek nagyon sok paraméterfile-t használnak a legkülönbözőbb formákban. Vannak például olyan segédfile-ok, amelyek tényleg csak paramétereket tartalmaznak, de vannak olyanok is, amelyekben az első rekord tartalmazza a paramétereket, a többi pedig adatterület. Léteznek soros vagy index-soros paraméterfile-ok, olyanok, amelyeket a program indításánál egyszer olvasunk be, vagy olyanok, amelyekből a feldolgozás során folyamatosan olvassuk a paramétereket, és így tovább.
Az alábbiakban a paraméterfile legegyszerűbb és legelterjedtebb típusát mutatjuk be.
A paramétereket sorosan tároljuk, és a program elején egyszerre olvassuk be.
A program indítás utáni első kérdése az, hogy be van-e kapcsolva a lemezegység, és a 8-as egységgel dolgozunk-e. Ha nem, akkor felszólít arra, hogy kapcsoljuk be a lemezegységet, és állítsuk 8-ra az egységszámot.
Ha az első kérdésre I-vel vagy i-vel válaszoltunk, akkor a számítógép felszólít arra, hogy helyezzünk be egy adatlemezt az egységbe. Az adatlemez lehet egészen új lemez is.
Ha új lemezt helyezünk a lemezegységbe, a program automatikusan megformálja. Ne feledkezzünk meg arról, hogy a formálás után a lemez korábbi tartalma elvész!
Pár másodperc múlva a program ismét visszajelentkezik és bekéri a paramétereket.
A képernyőn a következő képet látjuk:

```
Paraméterezett raktárnyilvántartás

értéktöbblet-adókulcs 1: ?7 <RETURN>

értéktöbblet-adókulcs 2: ?14 <RETURN>

Becsült tételszám (max. 3000): ? 500 <RETURN>
```

Az utolsó bevitel után kis időbe telik, amíg a program ismét bejelentkezik. Ez alatt az idő alatt a lemezegység dolgozik. Ebben nincs semmi rendellenes, hiszen a gép a lemezt először megformálja és ezután generálja a paraméter- és relatív file-t. Ha ez a folyamat befejeződött a képernyőn megjelenik a főmenü:

Paraméterezett raktárnyilvántartás

FŐMENÜ

1) A cikktörzs adatbevitele

2) A cikktörzs módosítása

3) A pénztárblokk bevitele

4) Raktárkezelés

5) A cikktörzs nyomtatása

6) A program vége

→ Kérem válasszon (1–6):

A megfelelő billentyű lenyomásával a különböző menüpontok közül kiválaszthatjuk a szükséges tevékenységet.

Az első szubrutin feltölti adatokkal a cikktörzs állományt. Ha például a „Hogyan leszek milliomos" című könyvet kívánjuk bekönyvelni, akkor a főmenüből az 1. pontot kell kiválasztanunk. Ekkor egy új képernyő jelenik meg, ahová beírhatjuk a megfelelő cikkszámot, a cikk megnevezését és végül az értéktöbblet-adócsoportot.
Az adatbevitel során a következőkre kell ügyelni: a cikkszám nem lehet nagyobb annál a számértéknél, amellyel a „cikkek darabszáma" kérdésre az előző pontban válaszoltunk.
A megnevezés a "," és a ":" karaktereket kivéve bármely karaktert tartalmazhat, de legfeljebb 20 karakter hosszúságú lehet. Az értéktöbblet-adócsoport kérdésre adott válasz egy 1 és 9 közé eső számérték lehet, de ezek közül az 1-es és 2-es értéktöbblet-adócsoporthoz rendeltünk a paraméterezésnél értelmes százalékértékeket. A többi csoportot a program 0 %-os értéktöbblet-adóval kezeli. Nézzünk egy példát az adatbevitelre:

Paraméterezett raktárnyilvántartás

A cikktörzs adatbevitele

1) Cikkszám: ? 125 <RETURN>

2) Megnevezés: ? Videokazetta <RETURN>

3) Készlet: ? 100 <RETURN>

Ha már a raktárban szereplő összes cikket begépeltük, cikkszám helyett gépeljük be a VEGE szöveget és nyomjuk le a RETURN billentyűt. A program lezárja a cikkfile-t és visszatér a főmenübe.

A második menüpont: "A cikktörzs módosítása" nagyon hasonlít az elsőhöz. Ezzel a programrésszel az eddig bevitt adatokat módosíthatjuk, vagy törölhetjük. Módosításhoz meg kell adni a cikkszámot, és a program válaszul kiírja a képernyőre a cikkszámhoz tartozó adatokat. Ekkor a megnevezést és az értéktöbblet-adókulcsot az eddigi bevitelek egyszerű átírásával javíthatjuk, illetve törölhetjük.
Ha egy mezőt nem akarunk megváltoztatni, egyszerűen nyomjuk le a RETURN billentyűt és a program megőrzi az eddigi adatot. A billentyűzeten van egy tréfásan "pókmajomnak" nevezett különleges karakter, a "p" és a "*" között. Ha egy cikk megnevezésének első karaktere helyére beírjuk ezt a karaktert, akkor ezt a cikket a program törli a nyilvántartásból. Valójában azonban ez a cikk fizikailag megmarad a file-ban, és ha a pókmajom karakter (@) helyére ismét mást írunk, újra használható lesz. A "módosító" programrészből ugyanúgy térhetünk vissza a főmenühöz, mint az "adatbevitel" programrészből.

A harmadik menüpontot választva bevihetjük a pénztárblokkokat, amelyeket a program azonnal el is könyvel és ezzel egyidejűleg a változtatást a nyomtatón jegyzőkönyvezi.
Felhívjuk az Olvasó figyelmét arra, hogy ez a program Commodore 1525 vagy Seikosha GP 100 VC nyomtatóra készült. Ha más nyomtatóval rendelkezünk, a programban néhány sort meg kell változtatni. A raktári kivételezést a cikkszám begépelésével kell kezdeni, mire válaszul a cikk adatai megjelennek a képernyőn. Ezt követően be kell írnunk az eladott darabszámot, a beszerzési árat és az eladási árat. A program automatikusan kiszámítja az összegeket, az eredményt kinyomtatja, és ezzel egyidejűleg a megadott darabszámot levonja a raktári készletből. Így mindig pontos áttekintésünk lesz az aktuális raktárkészletekről.

A főmenü következő pontja a raktárkezelés.
Itt is meg kell adni azt a cikkszámot, amelynél a raktári készlet változik. A képernyőn megjelenik a cikk megnevezése, az eddigi raktárkészlet és a módosításra vonatkozó kérdés. Ennek szemléltetésére lássunk egy példát:

Paraméterezett raktárnyilvántartás

Raktárkezelés

1) Cikkszám: 125

2) Megnevezés: Videokazetták

3) Raktárkészlet: 100

→ Módosítás: ?120

Ezzel a programrésszel könyvelhetjük el az áruszállítást. A fenti adatbevitel után az új készlet 220. Ha több módosítást nem akarunk végezni, a cikkszám helyett ismét a VEGE szöveget begépelve visszatérhetünk a menühöz.

A következő menüpont a teljes cikklista nyomtatására szolgál. A teljes listából pontos áttekintést kapunk a pillanatnyi raktárkészletekről, az adott cikkszámokról és az aktuális árakról. Íme egy példa arra, hogy hogyan nézhet ki egy cikklista:

DÁTUM: 1983. 11. 30.

ÜZLET: MINTAÜZLET, KEREPESTARCSA

Cikkszám	Megnevezés	Készlet	Vételár	Eladási ár
100	Üres kazetták	50	125	140
105	Üres hangszalagok	25	300	350
125	Videokazetták	225	600	670

Ha a főmenü utolsó pontját választjuk, a program befejezi a munkát és ezzel a napi feldolgozás befejeződött. Kivehetjük az adatlemezt az egységből és a gépet kikapcsolhatjuk.

2. RÉSZ: A PROGRAM ISMERTETÉSE

Az első sorok a keret-, a háttér- és a betűszín kiválasztására szolgálnak. A

```
POKE 53280, 14: REM KERETSZIN
POKE 53281, 14: REM HÁTTÉR
PRINT CHR$ (31): REM BETÜ
```

utasítások hatására világoskék alapon sötétkék betűket kapunk. A keret- és a háttérszín azonos választásával elkerülhetjük, hogy a keret megbontsa az egységes képet.
A következő programrész lemezműveleteket végez. Első lépésként inicializáljuk a lemezt OPEN 15,8,15,"IO" utasítással. Szükség esetén (új lemez) az OPEN 15,8,15,"N: ADATLEMEZ,64" utasítással megformáljuk a lemezt. Ekkor az OPEN 1,8,3,"O: ARTDAT,L" + CHR$ (48) utasítással azonnal előkészítjük a relatív file-t is. Az utasításban az L a relatív file-típusra utal, a programnév mögötti CHR$ karakter pedig a rekordok hosszúságát jelöli. Példánkban a rekordhossz 48 karakter. Egy rekord a következő mezőkből tevődik össze:

1. Cikkszám	= 4 karakter + RETURN
2. Megnevezés	= 20 karakter + RETURN
3. Raktárkészlet	= 4 karakter + RETURN
4. Értéktöbblet-adókulcs	= 1 karakter + RETURN
5. Beszerzési ár	= 7 karakter + RETURN
6. Eladási ár	= 7 karakter
Összeg	= 48 karakter

Ahhoz, hogy az egyes mezőket később a beolvasásnál és az írásnál különböző változókban tárolhassuk, egymástól RETURN karakterrel választottuk el.

A következő programrészben elkészítjük a paraméterfile-t. A paramétereket beolvassuk egymás után INPUT utasításokkal, majd felírjuk őket egy soros file-ba.
A file az 1. és 2. értéktöbblet-adókulcsot, és a cikkek maximális számát tartalmazza. Ez utóbbi többek között a relatív file elkészítéséhez szükséges.

```
OPEN 2,8,1,"O:PARAMETER"
PRINT#2, MS$ (1)
PRINT# 2, MS$ (2)
PRINT# 2, AN$
CLOSE 2
```

A fenti sorokkal felírjuk a paraméterfile-t a lemezre. Ezután a cikkek darabszámából kiszámítjuk a relatív file hosszát. A megadott értéket alsó és felső byte-ra bontjuk:

```
HB=INT (AR/256): LB=AR-HB*256
```

Ha a darabszám 999, az LB=231 a HB=3. A kiszámított értékekkel előkészítjük a relatív file-t. Erre szolgálnak a következő utasítások:

```
PRINT#15, "P"+CHR$ (3)+CHR$ (LB)+CHR$(HB)+CHR$(1)
PRINT# 1, CHR$ (255);
```

Az első utasítás egy parancsot küld a lemezegységhez és közli a DOS-szal, hogy a 3 másodlagos című adatcsatornán keresztül egy relatív file-t kell feldolgoznia, ami (a mi példánkban) 999 rekordból áll, továbbá hogy a feldolgozás mindig a rekordon belüli első karaktertől kezdődik. A második utasítással előkészítjük a file-t úgy, hogy az utolsó rekordba CHR$ (255) karaktert írunk. Az előkészítések után kezdődik a tényleges feldolgozás. Sajátosan oldottuk meg az adatok módosítását. A javítandó rekordot PRINT utasítással kiírjuk a képernyőre, és a CHR$ (145) karakterrel a kurzort a megfelelő sorra állítjuk. Ezt követően kényelmesen használhatjuk a képernyőszerkesztőt az INPUT utasításokkal megoldott beolvasáshoz.

Egy ötlet:
Ha a programban a kurzort a képernyő meghatározott helyére akarjuk mozgatni, akkor a következőt kell tenni:

```
HO$=CHR$ (19): CU$=CHR$ (17): CR$=CHR$ (29)
FOR I=1 TO 7
CU$=CU$+CU$: CR$=CR$+CR$
NEXT I
CU$=LEFT$ (CU$,25): CR$=LEFT$ (CR$,40)
```

Ha a fenti sorokat beillesztjük a programba, a pozicionálás egyetlen utasítással megoldható:

```
PRINT HO$; LEFT$ (CU$, VT); LEFT$ (CR$, HT);
```

ahol VT a sor, HT pedig az oszlop értékét tartalmazza.
Az adatok tárolásakor – mint már említettük – gondolnunk kell arra, hogy minden mező után egy RETURN karakter álljon. Erre szolgál a következő sor:

```
TE$=TE$ (1)+CR$+TE$(2)+CR$+TE$(3)+CR$+TE$(4)+CR$+TE$(5)+CR$+TE$(6)
PRINT#8, TE$
```

A munkát egy kissé megkönnyítettük azzal, hogy a program elején a CR$ változóba CHR$ (130) karaktert (RETURN) írtunk.
Az adatok visszaolvasása során csak arra kell ügyelni, hogy az olvasási sorrend azonos legyen a felírás sorrendjével:

```
INPUT#8, TE$(1), TE$(2), TE$(3), TE$(4), TE$(5), TE$(6)
```

Az egyes változók a következő mezőket tartalmazzák:

TE$ (1) = Cikkszám
TE$ (2) = Megnevezés
TE$ (3) = Raktárkészlet
TE$ (4) = Értéktöbblet-adókulcs
TE$ (5) = Beszerzési ár
TE$ (6) = Eladási ár

Az egyes mezők hosszúságát a program végén DATA sorokban tároltuk. Ha a mezők hossza az adott feladatban nem megfelelő, ezeket könnyen módosíthatjuk, de gondoljunk arra, hogy a relatív file készítésénél a hosszúság adatot meg kell változtatni [lásd fent – példa CHR$(48)]. Szeretnénk kitérni még néhány mondattal a hibacsatorna kezelésére. Az Olvasó feltehetően használta már a hibacsatornát, mégsem árt talán még egyszer összefoglalni a legfontosabb teendőket. A 0-ás és az 50-es hibát (0 = OK, 50 = RECORD NOT PRESENT) figyelmen kívül hagyhatjuk, mivel a rekordok lekérdezését a program automatikusan vezérli.

```
IF RN<1 OR RN>AN THEN...
```

RN ebben az esetben az elolvasandó/leírandó rekord, AN pedig a rekordok maximális száma. A fenti utasítással minden olyan rekordszámot kizárunk, amely a megadott paraméterek mellett érvénytelen.

P29

```
10000 REM *** PARAMETEREZETT RAKTARNYILVANTARTAS ***
10100 A=FRE(0):CLR
10200 DIMTE$(9),TD$(8),TI$(3),D1$(7),D2$(5),MS(3),MW(3)
10300 MS(3)=1
10400 CR$=CHR$(13)
10500 FORI=1TO8
10600 READTD$(I),TD(I)
10700 NEXT
10800 FORI=1TO3
10900 READTI$(I),TI(I)
11000 NEXT
11100 FORI=1TO7
11200 READD1$(I)
11300 NEXT
11400 FORI=1TO5
11500 READD2$(I)
11600 NEXT
11700 POKE53272,23:REM NAGYBETU/KISBETU
11800 POKE53280,14:REM A KERET SZINE
```

```
11900 POKE53281,14:REM A HATTER SZINE
12000 PRINTCHR$(31);:REM A BETUK SZINE
12100 GOSUB41300
12200 PRINT"A LEMEZEGYSEG BE VAN KAPCSOLVA (I/N) ?";
12300 GETEI$:IFEI$=""THEN12300
12400 IFEI$="I" OR EI$="I" THEN12600
12500 GOTO12300
12600 PRINTEI$:PRINT
12700 PRINT"AZ ADATLEMEZT HELYEZZE AZ EGYSEGBE!"
12800 GETEI$:IFEI$=""THEN12800
12900 PRINT:PRINT
13000 OPEN15,8,15,"I0"
13100 CLOSE15
13200 PRINT"UJ ADATLEMEZ (I/N)";
13300 GETEI$:IFEI$=""THEN13300
13400 IFEI$="N" OR EI$="N" THEN PRINT"N": GOTO 14500
13500 IFEI$="I" OR EI$="I" THENPRINT"I":GOTO13700
13600 GOTO13300
13700 OPEN15,8,15,"N0:ADATLEMEZ,64"
13800 GOSUB42100:GOSUB41300
13900 OPEN1,8,3,"0:CIKKTORZS,L,"+CHR$(48)
14000 PRINT#15,"P"+CHR$(3)+CHR$(LB)+CHR$(HB)+CHR$(1)
14100 PRINT#1,CHR$(255);
14200 RM=INT(167132/48)
14300 CLOSE1
14400 CLOSE15
14500 OPEN2,8,0,"0:PARAMETER"
14600 INPUT#2,MS$(1):MS(1)=VAL(MS$(1))
14700 INPUT#2,MS$(2):MS(2)=VAL(MS$(2))
14800 INPUT#2,AN$:AN=VAL(AN$)
14900 CLOSE2
15000 GOSUB41300
15100 PRINTTAB(15)"FOMENU":PRINT:PRINT
15200 PRINT"     1) A CIKKTORZS ADATBEVITELE":PRINT
15300 PRINT"     2) A CIKKTORZS MODOSITASA  ":PRINT
15400 PRINT"     3) PENZTARBLOKK BEVITELE":PRINT
15500 PRINT"     4) RAKTARKEZELES":PRINT
15600 PRINT"     5) A CIKKTORZS NYOMTATASA":PRINT
15700 PRINT"     6) A PROGRAM VEGE":PRINT:PRINT
15800 PRINT"-> KEREM VALASSZON (1-6): ";
15900 GETEI$:IFEI$=""THEN 15900
16000 WAHL=VAL(EI$)
16100 IFWAHL<1 OR WAHL>6 THEN 15900
16200 PRINTEI$
16300 FORI=1TO1000:NEXT
16400 ON WAHL GOTO 16600,19000,22600,32800,36600,40800
16500 END
16600 OPEN15,8,15
16700 OPEN8,8,8,"0:CIKKTORZS"
16800 GOSUB44900
16900 GOSUB41300
17000 PRINTTAB(7);"A CIKKTORZS ADATBEVITELE":PRINT:PRINT
17100 FORI=1TO3
17200 TE$(I)=""
17300 PRINTTI$(I);
17400 INPUTTE$(I)
17500 PRINT
17600 IFTE$(1)="VEGE" OR VAL(TE$(1))<1 THEN 18700
17700 IFLEN(TE$(I))>TI(I)THEN17200
17800 NEXT
17900 FORI=4TO6
18000 TE$(I)=""
18100 NEXT
18200 RN=VAL(TE$(1))
18300 IFRN<1ORRN>ANTHEN16900
18400 GOSUB44000
```

```
18500 GOSUB44600
18600 GOTO16900
18700 CLOSE8
18800 CLOSE15
18900 GOTO15000
19000 OPEN15,8,15
19100 OPEN8,8,8,"0:CIKKTORZS"
19200 GOSUB44900
19300 GOSUB41300
19400 PRINTTAB(8)"A CIKKTORZS MODOSITASA":PRINT:PRINT
19500 TE$(1)=""
19600 PRINTTI$(1);
19700 INPUTTE$(1)
19800 PRINT
19900 IFTE$(1)="VEGE" OR VAL(TE$(1))<1THEN22300
20000 IFLEN(TE$(1))>TI(1)THEN19500
20100 RN=VAL(TE$(1))
20200 IFRN<1ORRN>ANTHEN19300
20300 GOSUB44000
20400 GOSUB44400
20500 IFVAL(TE$(1))<>RNTHEN19300
20600 GOSUB41300
20700 PRINTTAB(8)"A CIKKTORZS MODOSITASA":PRINT:PRINT
20800 PRINTTI$(1);"  ";
20900 PRINTTE$(1)
21000 PRINT
21100 FORI=2TO3
21200 PRINTTI$(I);"? "
21300 PRINTTE$(I)
21400 PRINTCHR$(145);
21500 PRINTTI$(I);
21600 INPUTTE$(I)
21700 PRINT
21800 IFLEN(TE$(I))>TI(I)THEN21400
21900 NEXT
22000 GOSUB44000
22100 GOSUB44600
22200 GOTO19300
22300 CLOSE8
22400 CLOSE15
22500 GOTO15000
22600 OPEN15,8,15
22700 OPEN8,8,8,"0:CIKKTORZS"
22800 OPEN4,4,7
22900 GOSUB44900
23000 GOSUB41300
23100 GOSUB43700
23200 PRINT#4,"DATUM : ";DA$
23300 PRINT#4
23400 PRINT#4,"UZLET : ";GE$
23500 PRINT#4
23600 FORI=1TO79
23700 PRINT#4,"-";
23800 NEXT
23900 PRINT#4
24000 PRINT#4,CHR$(16)"01"D1$(1);
24100 PRINT#4,CHR$(16)"10"D1$(2);
24200 PRINT#4,CHR$(16)"30"D1$(3);
24300 PRINT#4,CHR$(16)"40"D1$(4);
24400 PRINT#4,CHR$(16)"50"D1$(5);
24500 PRINT#4,CHR$(16)"60"D1$(6);
24600 PRINT#4,CHR$(16)"70"D1$(7)
24700 FORI=1TO79
24800 PRINT#4,"-";
24900 NEXT
25000 PRINT#4
```

```
25100 GOSUB41300
25200 PRINTTAB(7)"A PENZTARBLOKK BEVITELE":PRINT:PRINT
25300 TE$(1)=""
25400 PRINTTD$(1);
25500 INPUTTE$(1)
25600 PRINT
25700 IFTE$(1)="VEGE" OR VAL(TE$(1))<1THEN31100
25800 IFLEN(TE$(1))>TD(1)THEN25300
25900 RN=VAL(TE$(1))
26000 IFRN<1ORRN>ANTHEN25100
26100 GOSUB44000
26200 GOSUB44400
26300 IFVAL(TE$(1))<>RNTHEN25100
26400 GOSUB41300
26500 PRINTTAB(7)"A PENZTARBLOKK BEVITELE":PRINT:PRINT
26600 PRINTTD$(1);"  ";
26700 PRINTTE$(1)
26800 PRINT
26900 PRINTTD$(2);"  ";
27000 PRINTTE$(2)
27100 PRINT
27200 PRINTTD$(4);
27300 INPUTTE$(7)
27400 PRINT
27500 IFLEN(TE$(7))>TD(4)THEN27200
27600 PRINTTD$(5)"? ";
27700 PRINTTE$(5)
27800 PRINTCHR$(145);
27900 PRINTTD$(5);
28000 INPUTTE$(5)
28100 PRINT
28200 IFLEN(TE$(5))>TD(5)THEN27900
28300 PRINTTD$(7)"? ";
28400 PRINTTE$(6)
28500 PRINTCHR$(145);
28600 PRINTTD$(7);
28700 INPUTTE$(6)
28800 PRINT
28900 IFLEN(TE$(6))>TD(7)THEN28600
29000 TE$(4)=STR$(VAL(TE$(4))-VAL(TE$(7)))
29100 IFVAL(TE$(4))<-999 THENTE$(4)="-999"
29200 TE$(8)=STR$(VAL(TE$(5))*VAL(TE$(7)))
29300 TE$(9)=STR$(VAL(TE$(6))*VAL(TE$(7)))
29400 EE=EE+VAL(TE$(5)):EE$=STR$(EE)
29500 GE=GE+VAL(TE$(8)):GE$=STR$(GE)
29600 EV=EV+VAL(TE$(6)):EV$=STR$(EV)
29700 GV=GV+VAL(TE$(9)):GV$=STR$(GV)
29800 GR=VAL(TE$(3)):WE=VAL(TE$(9))
29900 IFGR<>1 AND GR<>2THENGR=3
30000 MW(GR)=MW(GR)+WE*MS(GR)/100
30100 GOSUB44000
30200 GOSUB44600
30300 PRINT#4,CHR$(16)"01"TE$(1);
30400 PRINT#4,CHR$(16)"10"TE$(2);
30500 PRINT#4,CHR$(16)"35"TE$(7);
30600 PRINT#4,CHR$(16)"40"TE$(5);
30700 PRINT#4,CHR$(16)"50"TE$(8);
30800 PRINT#4,CHR$(16)"60"TE$(6);
30900 PRINT#4,CHR$(16)"70"TE$(9)
31000 GOTO 25100
31100 FORI=1TO79
31200 PRINT#4,"-";
31300 NEXT
31400 PRINT#4
31500 PRINT#4,CHR$(16)"01";":";
31600 PRINT#4,CHR$(16)"40"EL$;:EL=0
```

```
31700 PRINT#4,CHR$(16)"50"GE$;:GE=0
31800 PRINT#4,CHR$(16)"60"EV$;:EV=0
31900 PRINT#4,CHR$(16)"70"GV$;:GV=0
32000 PRINT#4
32100 PRINT#4,CHR$(16)"01RT. ADO"MS(1)"% = "MW(1)
32200 PRINT#4,CHR$(16)"01RT. ADO"MS(2)"% = "MW(2)
32300 PRINT#4,CHR$(16)"01RT. ADO 0 % ="MW(3)
32400 CLOSE4
32500 CLOSE8
32600 CLOSE15
32700 GOTO 15000
32800 OPEN15,8,15
32900 OPEN8,8,8,"0:CIKKTORZS"
33000 GOSUB44900
33100 GOSUB41300
33200 PRINTTAB(14)"RAKTARKEZELES":PRINT:PRINT
33300 TE$(1)=""
33400 PRINTTD$(1);
33500 INPUTTE$(1)
33600 PRINT
33700 IFTE$(1)="VEGE" OR VAL(TE$(1))<1THEN36300
33800 IFLEN(TE$(1))>TD(1)THEN33300
33900 RN=VAL(TE$(1))
34000 IFRN<1 OR RN>AN THEN 33100
34100 GOSUB44000
34200 GOSUB44400
34300 IFVAL(TE$(1))<>RN THEN33100
34400 GOSUB41300
34500 PRINTTAB(14)"RAKTARKEZELES":PRINT:PRINT
34600 PRINTTD$(1)"  ";
34700 PRINTTE$(1)
34800 PRINT
34900 PRINTTD$(2)"  ";
35000 PRINTTE$(2)
35100 PRINT
35200 PRINTTD$(3)"  ";
35300 PRINTTE$(4)
35400 PRINT
35500 PRINT"-> RAKTARKEZELES  :";
35600 INPUTTE$(7)
35700 IFLEN(TE$(7))>TD(4)THEN35500
35800 TE$(4)=STR$(VAL(TE$(4))+VAL(TE$(7)))
35900 IFVAL(TE$(4))>9999THENTE$(4)="9999"
36000 GOSUB44000
36100 GOSUB44600
36200 GOSUB33100
36300 CLOSE8
36400 CLOSE15
36500 GOTO15000
36600 OPEN15,8,15
36700 OPEN8,8,8,"0:CIKKTORZS"
36800 OPEN4,4,7
36900 GOSUB44900
37000 GOSUB41300
37100 GOSUB43700
37200 PRINT#4,"UZLET : ";DA$
37300 PRINT#4
37400 PRINT#4," : "GE$
37500 PRINT#4
37600 FORI=1TO79
37700 PRINT#4,"-";
37800 NEXT
37900 PRINT#4
38000 PRINT#4,CHR$(16)"01"D2$(1);
38100 PRINT#4,CHR$(16)"10"D2$(2);
38200 PRINT#4,CHR$(16)"35"D2$(3);
```

```
38300 PRINT#4,CHR$(16)"40"D2$(4);
38400 PRINT#4,CHR$(16)"60"D2$(5)
38500 FORI=1TO79
38600 PRINT#4,"-";
38700 NEXT
38800 PRINT#4
38900 FORRN=1TOAN
39000 GOSUB44000
39100 GOSUB44400
39200 IFVAL(TE$(1))<>RN THEN 39900
39300 IFLEFT$(TE$(2),1)="&"THEN39900
39400 PRINT#4,CHR$(16)"01"TE$(1);
39500 PRINT#4,CHR$(16)"10"TE$(2);
39600 PRINT#4,CHR$(16)"35"TE$(4);
39700 PRINT#4,CHR$(16)"40"TE$(5);
39800 PRINT#4,CHR$(16)"60"TE$(6)
39900 NEXT
40000 FORI=1TO79
40100 PRINT#4,"-";
40200 NEXT
40300 PRINT#4
40400 CLOSE4
40500 CLOSE8
40600 CLOSE15
40700 GOTO15000
40800 CLOSE4
40900 CLOSE8
41000 CLOSE15
41100 PRINTCHR$(147)
41200 END
41300 PRINTCHR$(147);
41400 PRINT"*****************************************";
41500 PRINT"*                                       *";
41600 PRINT"*  PARAMETEREZETT RAKTARNYILVANTARTAS    *";
41700 PRINT"*                                       *";
41800 PRINT"*****************************************";
41900 PRINT:PRINT
42000 RETURN
42100 GOSUB41300
42200 INPUT"ADO KULCS  1 : ";MS$(1):MS(1)=VAL(MS$(1)):PRINT
42300 IFVAL(MS$(1))<1 OR VAL(MS$(1))>50 THEN42200
42400 INPUT"ADO KULCS  2 : ";MS$(2):MS(2)=VAL(MS$(2)):PRINT
42500 IFVAL(MS$(2))<1 OR VAL(MS$(2))>50 THEN42400
42600 INPUT"BECSULT TETELSZAM  (10 - 3000) : ";AN$:PRINT
42700 IFVAL(AN$)<10 OR VAL(AN$)>3000 THEN 42600
42800 AR=VAL(AN$)
42900 HB=INT(AR/256)
43000 LB=AR-256*HB
43100 OPEN2,8,1,"0:PARAMETER"
43200 PRINT#2,MS$(1)
43300 PRINT#2,MS$(2)
43400 PRINT#2,AN$
43500 CLOSE2
43600 RETURN
43700 INPUT"A MAI DATUM  (EE.HH.NN) : ";DA$:PRINT
43800 INPUT"UZLET: ";GE$:PRINT
43900 RETURN
44000 HB=INT(RN/256):LB=RN-HB*256
44100 PRINT#15,"P"CHR$(8)CHR$(LB)CHR$(HB)CHR$(1)
44200 GOSUB44900
44300 RETURN
44400 INPUT#8,TE$(1),TE$(2),TE$(3),TE$(4),TE$(5),TE$(6)
44500 RETURN
44600 TE$=TE$(1)+CR$+TE$(2)+CR$+TE$(3)+CR$+TE$(4)+CR$+TE$(5)+CR$+TE$(6)
44700 PRINT#8,TE$
44800 RETURN
```

```
44900 INPUT#15,F1$,F2$,F3$,F4$
45000 IFVAL(F1$)=0ORVAL(F1$)=50THENRETURN
45100 PRINT"FEHLER: "F1$" "F2$" "F3$" "F4$
45200 CLOSE4:CLOSE8
45300 CLOSE15
45400 FORI=1TO6000:NEXT
45500 GOTO12100
45600 DATA"1) CIKKSZAM       :",4
45700 DATA"2) MEGNEVEZES      :",20
45800 DATA"3) KESZLET         :",4
45900 DATA"4) MENNYISEG       :",3
46000 DATA"5) BESZERZESI AR   :",7
46100 DATA"6) BESZERZESI ERTEK:",8
46200 DATA"7) ELADASI AR      :",7
46300 DATA"8) ELADAS OSSZESEN :",8
46400 DATA"1) CIKKSZAM        :",4
46500 DATA"2) MEGNEVEZES      :",20
46600 DATA"3) ERTEK ADO       :",1
46700 DATA"SSZ."
46800 DATA"MEGNEVEZES"
46900 DATA"KESZLET"
47000 DATA"BESZ. AR"
47100 DATA"BESZ. ERT"
47200 DATA"ELAD. AR"
47300 DATA"ELAD. ERT"
47400 DATA"SSZ."
47500 DATA"MEGNEVEZES"
47600 DATA"KESZLET"
47700 DATA"BESZ. AR"
47800 DATA"ELAD. AR"

READY.
```

A program változói:

A	segédváltozó
TE$ ( )	mezőtartalom (1–9)
TD$ ( )	1. fejléc
TD ( )	1. mezőhossz
TI$ ( )	2. fejléc
TI ( )	2. mezőhossz
D1$ ( )	nyomtató 1. felirat
D2$ ( )	nyomtató 2. felirat
MS ( )	értéktöbblet-adókulcsok
MW ( )	értéktöbblet-adók összege
CR$	segédváltozó (=RETURN)
I	ciklusváltozó
EI$	változó egy karakter beolvasásához
LB	a rekordsorszám alsó byte-ja relatív file kezelésénél
HB	a rekordsorszám felső byte-ja relatív file kezelésénél
RM	a lehetséges rekordok maximális száma
AN	a tényleges rekordok maximális száma
WAHL	a választott menüpont
RN	olvasandó/írandó rekord
EE	beszerzési árak összege
GE	beszerzési ár összesen
EV	eladási árak összege

GV	eladás összesen
GR	választott értéktöbblet-adócsoport
WE	értéktöbblet-adó összege
EE$	lásd EE
GE$	lásd GE
EV$	lásd EV
GV$	lásd GV

4.3 Egy általános listázó

1. RÉSZ:

Ezzel a programmal egy tetszőleges soros file tartalmát kinyomtathatjuk. A programot használhatjuk olyan relatív file listázására is, amelyben az adatmezőket RETURN karakter választja el egymástól. A nyomtatáshoz szükség van a file-ra vonatkozó néhány információra. A programmal tudatni kell pl., hogy hány mező található egy rekordon belül és mik az egyes mezők megnevezései. A program kezelése rendkívül egyszerű. Meg kell adni a listázandó file nevét, majd a rekordonkénti mezőszámot.
Ezután a program bekéri a nyomtatandó mezők sorszámát és hosszúságát. Az egyes mezők hosszúsága nem haladhatja meg a 80 karaktert. Fontos tudni a program egy sajátosságát: ha egy mező megnevezése pl. az, hogy POSTAI IRÁNYÍTÓSZÁM, akkor annak ellenére, hogy a mező hosszúsága 4 karakter, a program 19 karaktert foglal le, mivel a POSTAI IRÁNYÍTÓSZÁM (szöveg) 19 karakter hosszúságú és ezt szintén ki kell nyomtatni.
A fenti paraméterek után a program egyenként beolvassa a rekordokat a file-ból és ugyanabban a sorrendben kiírja azokat a nyomtatóra. Az utolsó rekord kiírása után választhatunk, hogy egy újabb file-t kívánunk kinyomtatni vagy befejezzük a munkát.

2. RÉSZ:

A program pontosan olyan egyszerű, mint a kezelése. Az sem jelent különösebb programozási nehézséget, hogy a relatív file feldolgozását is lehetővé tettük.
Első lépésként megnyitjuk a file-t olvasásra:

```
OPEN 8,8,8,DN$+",S,R"
```

ahol a DN$ változóban a kiírandó file nevét tároljuk. A program további része egyetlen ciklus, amelyben minden rekordot egymás után beolvasunk a tárba, és kiválasztjuk belőlük azokat a mezőket, amelyeket ki kell nyomtatni. Ez a művelet a file végének (END OF FILE) érzékeléséig ismétlődik.
Befejezésként a file-okat (a nyomtatandó és a nyomtató file-t) lezárjuk.

```
P30

10000 REM ALTALANOS LISTAZO
10100 A=FRE(0):CLR
10200 DIM TI$(99),FE$(99),TE$(99)
10300 POKE53272,23:REM KISBETU/NAGYBETU
10400 POKE53280,14:REM A KERET SZINE
10500 POKE53281,14:REM A HATTER SZINE
10600 PRINT CHR$(31):REM A BETUK SZINE
10700 PRINT CHR$(147);
10800 PRINT"*** ALTALANOS LISTAZO ***":PRINT:PRINT
10900 PRINT"-> MAI DATUM (EE.HH.NN) :";
11000 INPUT DA$
11100 PRINT CHR$(147);
11200 PRINT"*** ALTALANOS LISTAZO ***":PRINT:PRINT
11300 PRINT"-> A FILE NEVE :";
11400 INPUT DN$: PRINT
11500 IF DN$="" THEN 16800
11600 OPEN 15,8,15
11700 OPEN 8,8,8,"0:"+DN$
```

```
11800 OPEN 4,4,7
11900 GOSUB 17300
12000 PRINT"-> A MEZOK SZAMA :";
12100 INPUT AF$:PRINT
12200 IF VAL(AF$)<1 OR VAL(AF$)>99 THEN 12000
12300 AF=VAL(AF$)
12400 PRINT"-> NYOMTATASI MEZOK SZAMA :";
12500 INPUT AD$:PRINT
12600 IF VAL(AD$)<1 OR VAL(AD$)>AF THEN 12400
12700 AD=VAL(AD$)
12800 FOR I=1 TO AD
12900 PRINT"-> CIM #";I;" :";
13000 INPUT TI$(I)
13100 IF LEN(TI$(I))>20 THEN TI$(I)=LEFT$(TI$(I),20)
13200 PRINT"-> MEZOSZAM :";
13300 INPUT FE$(I)
13400 PRINT
13500 IF VAL(FE$(I))<1 OR VAL(FE$(I))>AF THEN 12900
13600 NEXT
13700 FOR I=1 TO 79
13800 PRINT#4,"-";
13900 NEXT
14000 PRINT#4
14100 PRINT#4,"DATUM : ";DA$
14200 PRINT#4
14300 PRINT#4,"FILENEV : ";DN$
14400 FOR I=1 TO 79
14500 PRINT#4,"-";
14600 NEXT
14700 PRINT#4
14800 FOR I=1 TO AF
14900 INPUT#8,TE$(I)
15000 NEXT
15100 IF ST AND 64 THEN EOF=1
15200 FOR I=1 TO AD
15300 PRINT#4,CHR$(16);"01";TI$(I);
15400 PRINT#4,CHR$(16);"22";":";
15500 PRINT#4,CHR$(16);"25";LEFT$(TE$(VAL(FE$(I))),54)
15600 NEXT
15700 PRINT#4
15800 IF EOF THEN 16000
15900 GOTO 14800
16000 FOR I=1 TO 79
16100 PRINT#4,"-";
16200 NEXT
16300 PRINT#4
16400 CLOSE4
16500 CLOSE8
16600 CLOSE15
16700 GOTO 11100
16800 CLOSE4
16900 CLOSE8
17000 CLOSE15
17100 PRINT CHR$(147)
17200 END
17300 INPUT#15, F1$,F2$,F3$,F4$
17400 IF VAL(F1$)=0 THEN RETURN
17500 PRINT "HIBA: ";F1$;" ";F2$;" ";F3$;" ";F4$
17600 CLOSE4
17700 CLOSE8
17800 CLOSE15
17900 FOR I=1 TO 6000:NEXT
18000 GOTO11100

READY.
```

A program változói:

A	segédváltozó
TI$ ( )	a mező neve
TE$ ( )	a mező sorszáma
TE$ ( )	a mező tartalma
AF$	a mezők száma
AF	lásd AF$
AD$	a nyomtatandó mezők száma
AD	lásd AD$
I	ciklusváltozó
EOF	mutató, mely a file végét jelzi.

4.4 Egy egyszerű szövegfeldolgozó program

Minden számítógéphez kapható szövegfeldolgozó program. De sajnos kevés kivételtől eltekintve a szövegfeldolgozó programok nagyon drágák és ráadásul többnyire szűk területen alkalmazhatóak. Most bemutatunk egy kis szövegfeldolgozó programot, ami ugyan messze nem olyan teljesítményű, mint a TEXTOMAT, a Commodore 64-es DATA BECKER szövegfeldolgozó rendszere, de kisebb szövegek kezelésére kiválóan alkalmas és megkímélheti az olvasót a kisebb írási munkáktól.
Indítás után a képernyőn a következő főmenü jelentkezik:

Szövegfeldolgozás

1) szövegbevitel billentyűzetről

2) szövegmódosítás billentyűzetről

3) szövegkiírás a képernyőre

4) szövegkiírás a nyomtatóra

5) a szöveg tárolása

6) a szöveg törlése

7) a program vége

Ha lenyomjuk az "1" billentyűt, a program megkérdezi:

..... sorszámtól :?

Válaszként megadhatjuk azt a sorszámot, ahol a szöveg bevitelét szeretnénk elkezdeni. Ha ugyanabba a sorba egyszer már írtunk szöveget, az új bevitelnél a régi sor elvész. A többi sort ez a bevitel nem érinti.
Egy sorba maximum 79 karaktert írhatunk. Ha ezt elértük, a program automatikusan sort emel. Célszerű tehát a szöveg bevitelénél arra ügyelni, hogy idejében sort váltsunk (RETURN). A képernyőn megjelenik a következő sor, és folytathatjuk a szöveget. Maximálisan 99 sort írhatunk le. Ha ezt elértük, vagy megnyomjuk az F1 billentyűt, a program visszatér a főmenübe.

A "2" billentyű lenyomásával korrigálhatjuk a kész szöveget. A számítógép megkérdezi:

Melyik sorszám :?

Most beírhatjuk annak a sornak a számát, amit módosítani szeretnénk. A számítógép kiírja a sorszámot és magát a sort, ekkor begépelhetjük a javítást. A RETURN billentyű megnyomása után a program ismét sorszámot kér. Ha válasz helyett ismét a RETURN billentyűt nyomjuk le, a program visszatér a főmenühöz.
Ha javítás közben a kurzort jobbra mozgatjuk, a program a régi sort új sorba másolja. A jobbra mutató nyílbillentyű hatására pedig a kurzortól balra lévő karakterek törlődnek (csak az új sorban!). Ha az új sor első karaktereként a RETURN billentyűt nyomjuk le, a teljes

régi sor áthelyeződik az új sorba. Egyébként az új sor beolvasása befejeződik a RETURN hatására, és a szövegben a megfelelő helyre kerül.

Ha ki akarjuk íratni a képernyőre a teljes szöveget, a főmenüből a "szövegkiírás a képernyőre" pontot kell választanunk. Pár másodperc múlva a teljes szöveg megjelenik a képernyőn az 1-es sorszámtól kezdve. A szóköz billentyű lenyomásával a képernyőt megállíthatjuk és ismét továbbfuttathatjuk. A szöveg teljes kiírását a program a

* VÉGE *

szó kiírásával jelzi.
A képernyő tartalma azonban mindaddig változatlan, míg ismét le nem nyomjuk a szóköz billentyűt, ekkor a program visszatér a főmenübe.

A szöveget a nyomtatóra is kiírathatjuk, a 4-es pont választásával. A program felszólítja a gépkezelőt, hogy kapcsolja be a nyomtatót, állítsa be a papírt és a RETURN billentyű lenyomásával jelezze, ha készen van.

A nyomtatás végén a program ismét a főmenübe tér vissza.
Az 5. és 6. pontok önmagukért beszélnek. A teljes szöveget tárolhatjuk lemezen és betölthetjük a tárba.
A program befejezéséhez nyomjuk le a főmenüben a 7-es billentyűt.

2. RÉSZ:

A szövegfeldolgozásnál nagyon fontos a billentyűzet állandó figyelése. A billentyűk lekérdezését az operációs rendszer input rutinja nem biztosítja. A Commodore 64 BASIC nyelvében a GET utasítást használhatjuk erre a célra. A GET utasítással karakterenként olvashatunk a klaviatúráról és minden lépésben megvizsgálhatjuk, hogy milyen billentyűt nyomott le a gépkezelő.
Egy egyszerű rutinnal a teljes szövegbevitel ellenőrizhető:

```
100 GET EI$: IF EI$ = " "THEN 100
110 IF EI$>CHR$(31)AND EI$<CHR$(128)THEN 100
```

A rutin csak az úgynevezett ASCII tartományban lévő karaktereket fogadja el, azaz minden különleges és vezérlőkaraktert figyelmen kívül hagy. A GET utasítás azonban másutt is segít. Ha ismét be akarunk olvasni egy szöveget, előfordulhat, hogy a szöveg egy része időközben elveszett. Ennek az a magyarázata, hogy a normál INPUT utasítás csak a vesszőig (,) olvas. Mivel egy szövegben nagyon gyakran előfordulhatnak vesszők, gondoskodnunk kell arról, hogy a mi beolvasó rutinunk ezt a karaktert elfogadja.

```
P31

100 OPEN 1,8,0,DN$;",S,R"
110 GET#1,EI$
120 TE$=TE$+EI$
130 IF EI$=CHR$(13) OR LEN(TE$)=80 THEN 150
140 GOTO 110
150 PRINT TE$

READY.
```

```
10000 REM SZOVEGFELDOLGOZAS
10100 A=FRE(0):CLR
10200 DIM TE$(100)
10300 POKE 53272,23:REM KISBETU/NAGYBETU
10400 POKE 53280,14:REM A KERET SZINE
10500 POKE 53281,14:REM A HATTER SZINE
10600 PRINT CHR$(31):REM A BETUK SZINE
10700 PRINT CHR$(147);
10800 PRINT "*****************************"
10900 PRINT "*                           *"
11000 PRINT "*    SZOVEGFELDOLGOZAS      *"
11100 PRINT "*                           *"
11200 PRINT "*****************************"
11300 PRINT:PRINT
11400 PRINTTAB(15);"FOMENU: ":PRINT:PRINT
11500 PRINT"    1)  SZOVEGBEVITEL BILLENTYUZETROL":PRINT
11600 PRINT"    2)  SZOVEGMODOSITAS BILLENTYUZETROL":PRINT
11700 PRINT"    3)  SZOVEGKIIRAS A KEPERNYORE":PRINT
11800 PRINT"    4)  SZOVEGKIIRAS A NYOMTATORA":PRINT
11900 PRINT"    5)  A SZOVEG TAROLASA":PRINT
12000 PRINT"    6)  A SZOVEG BETOLTESE":PRINT
12100 PRINT"    7)  A PROGRAM VEGE":PRINT
12200 PRINT"-> KEREM VALASSZON! (1-7) : ";
12300 GET EI$:IFEI$="" THEN 12300
12400 WAHL=VAL(EI$)
12500 IF WALH<1 OR WALH>7 THEN 12300
12600 ON WALH GOTO 12800,16000,19600,21300,22200,23200,24600
12700 END
12800 PRINT CHR$(147);
12900 PRINT "-> SORSZAMTOL :";
13000 INPUT ZE$
13100 PRINT
13200 IF VAL(ZE$)<1 OR VAL(ZE$)>99 THEN 12900
13300 ZE=INT(VAL(ZE$))
13400 PRINT
13500 PRINT ZE;": ";
13600 AZ=0
13700 GET EI$
13800 PRINT CHR$(185);CHR$(157);
13900 IF EI$="" THEN 13700
14000 IF EI$<>CHR$(20) THEN 14600
14100 IF LEN(TE$(ZE))<1 THEN 13700
14200 PRINT " ";CHR$(157);
14300 PRINT CHR$(157);" ";CHR$(157);
14400 TE$(ZE)=LEFT$(TE$(ZE),LEN(TE$(ZE))-1)
14500 GOTO 13700
14600 IF EI$<>CHR$(133) THEN 14900
14700 IF LEN(TE$(ZE))<1 THEN 15700
14800 GOTO 13700
14900 IF AZ=0 THEN TE$(ZE)="": AZ=1
15000 PRINT " ";CHR$(157);EI$;
15100 IF EI$=CHR$(13) THEN ZE=ZE+1: GOTO 15400
15200 TE$(ZE)=TE$(ZE)+EI$
15300 IF LEN(TE$(ZE))=79 THEN TE$(ZE)=TE$(ZE)+CHR$(13): ZE=ZE+1
15400 IF ZE=100 THEN 15700
15500 IF EI$=CHR$(13) THEN 13500
15600 GOTO 13700
15700 PRINT " ";CHR$(157)
15800 TE$(ZE)="!!"
15900 GOTO 10700
16000 PRINT CHR$(147);
16100 PRINT "-> SORSZAM :";
16200 INPUT ZE$
16300 PRINT
```

```
16400 IF VAL(ZE$)<1 OR VAL(ZE$)>99 THEN 16100
16500 ZE=INT(VAL(ZE$))
16600 PRINT
16700 PRINT ZE;": ";
16800 PRINT TE$(ZE)
16900 PRINT
17000 PRINT ZE;": ";
17100 NT$=""
17200 GET EI$
17300 PRINT CHR$(185);CHR$(157);
17400 IF EI$="" THEN 17200
17500 IF EI$<>CHR$(20) THEN 18100
17600 IF LEN(NT$)<1 THEN 17200
17700 PRINT " ";CHR$(157);
17800 PRINT CHR$(157);" ";CHR$(157);
17900 NT$=LEFT$(NT$,LEN(NT$)-1)
18000 GOTO 17200
18100 IF  EI$<>CHR$(29) THEN 18400
18200 IFLEN(NT$)<=LEN(TE$(ZE)) THEN EI$=MID$(TE$(ZE),LEN(NT$)+1,1):GOTO 18800
18300 GOTO 17200
18400 IF EI$<>CHR$(133) THEN 18800
18500 IF LEN(NT$)<1 THEN 10700
18600 GOTO 17200
18700 IF AZ=0 THEN NT$="": AZ=1
18800 PRINT " ";CHR$(157);EI$;
18900 IF EI$=CHR$(13) THEN 19300
19000 NT$=NT$+EI$
19100 IF LEN(NT$)=79 THEN NT$=NT$+CHR$(13)
19200 GOTO 17200
19300 IF LEN(NT$)<1 THEN NT$=TE$(ZE): PRINT NT$
19400 TE$(ZE)=NT$
19500 GOTO 10700
19600 PRINT CHR$(147);
19700 ZE=1
19800 IF LEFT$(TE$(ZE),2)="!!" THEN20400
19900 PRINT ZE;": ";
20000 PRINT TE$(ZE)
20100 GOSUB 20800
20200 ZE=ZE+1: IF ZE=100 THEN 20400
20300 GOTO 19800
20400 PRINT
20500 PRINT "** VEGE **";
20600 GET EI$: IF EI$="" THEN 20600
20700 GOTO 10700
20800 GET IE$
20900 IF EI$<>CHR$(32) THEN RETURN
21000 GET EI$
21100 IF EI$<>CHR$(32) THEN 21000
21200 RETURN
21300 OPEN 4,4,7
21400 ZE=1
21500 IF LEFT$(TE$(ZE),2)="!!" THEN 22000
21600 PRINT#4,TE$(ZE)
21700 ZE=ZE+1
21800 IF ZE=100 THEN 22000
21900 GOTO 21500
22000 CLOSE 4
22100 GOTO 10700
22200 PRINT CHR$(147);
22300 PRINT "-> SZOVEGNEV :";
22400 INPUT DN$
22500 OPEN 1,8,2,"@0:"+DN$+",S,W"
22600 FOR ZE=1 TO 100
22700 IF TE$(ZE)="" THEN TE$(ZE)=CHR$(13)
22800 PRINT#1, TE$(ZE)
22900 NEXT
```

```
23000 CLOSE1
23100 GOTO 10700
23200 PRINT CHR$(147);
23300 PRINT "-> SZOVEGNEV :";
23400 INPUT DN$
23500 OPEN 1,8,0,DN$+",S,R"
23600 FOR ZE=1 TO 100
23700 TE$=""
23800 GET#1,EI$
23900 TE$=TE$+EI$
24000 IF EI$=CHR$(13) OR EI$="" THEN 24200
24100 GOTO 23800
24200 TE$(ZE)=TE$
24300 NEXT
24400 CLOSE 1
24500 GOTO 10700
24600 PRINT CHR$(147)
24700 END

READY.
```

A program változói:

A	segédváltozó
TE$ ( )	1–99 szövegsor
WAHL	a választott menüpont
I	ciklusváltozó
ZE$	sorszám
ZE	lásd ZE$
AZ	segédváltozó
EI$	változó egy karakter beolvasásához
NT$	a módosított sor
DN$	a file neve

4.5 A QUISAM alkalmazása – irodalomjegyzékek nyilvántartása

A könyv 2. fejezetében elmondtunk mindent, amit a file-szervezésről egy jó program elkészítéséhez tudni kell. A hasznos segédprogramok egész sorát bocsátottuk az Olvasó rendelkezésére. Úgy gondoljuk azonban, hogy a témakört teljesen csak egy gyakorlati alkalmazás ismertetésével zárhatjuk le megnyugtatóan.
Most bemutatunk egy olyan programot, amelyben felhasználtuk a QUISAM nyújtotta összes lehetőséget. Ha az Olvasó a számítógépes folyóiratok buzgó olvasója, bizonyára előfordult már vele, hogy miközben egy meghatározott témáról olvasott egy cikket, halványan derengeni kezdett, hogy: "Várjunk csak, erről már olvastam valamit, hogy is volt?" és ami még rosszabb, az sehogy se jutott eszébe, hogy: HOL OLVASTAM?
Ezt a problémát egyszer és mindenkorra kiküszöbölhetjük. Olyan irodalomjegyzék-kezelő programot írunk, amely lehetővé teszi számunkra, hogy a témakörök alapján kikeressük a cikkeket és közli velünk, hogy eddig az adott témával kapcsolatban mikor és melyik folyóiratban jelentek meg cikkek.
Első pillantásra úgy tűnik, hogy a feladatban nincs semmi rendkívüli. De ha jól átgondoljuk, hogy mit is akarunk a programban megoldani, gyorsan kiderül, hogy a megoldás nem is olyan egyszerű. Egy normál relatív file azonos hosszúságú rekordokkal dolgozik, ezen alapul a közvetlen visszakeresés lehetősége. A mi feladatunkban azonban a téma címének hosszúságát ugyan rögzíthetjük, de hogy ezzel a témával kapcsolatban hány helyen jelent meg cikk, azt előre nem tudhatjuk.
Minden új folyóirattal növekedhet ez a szám. Előre tervezünk mondjuk 20 hivatkozást. És azután mi lesz?
De még ha el is tekintünk a további hivatkozások tárolásától, akkor is rossz a megoldás, mert helypazarló.
Hiszen az egyik témához többet, a másikhoz kevesebbet írunk. Miért tervezzünk hát minden téma számára ugyanannyi helyet? A feladatot csak dinamikusan növekvő rekordhosszal felépített file segítségével oldhatjuk meg, ami nem más, mint a QUISAM. A megjegyzés file lehetővé teszi számunkra, hogy tetszőlegesen sok hivatkozást tároljunk témánként és mindig csak annyi helyet foglalunk el, amennyire ténylegesen szükségünk van.

Mit kell tehát tudnia a programnak?

A file előkészítése
A téma bevitele
A hivatkozás bevitele
A téma keresése
A téma törlése
A téma nyomtatása

A "téma nyomtatása" azt jelenti, hogy egy témához ki kell nyomtatni a témára vonatkozó összes hivatkozást.

Mit akarunk tárolni?

Pontosan két beviteli mezőre van szükségünk:
Téma
Hivatkozás

A téma tartalmazza a témakört, a hivatkozás a folyóirat nevét, a megjelenési dátumot (hónapban, évben) ill. az oldalszámot. A témakörök megválasztását teljesen az Olvasóra bízzuk.
A törzsrekord tehát egyetlen mezőből, a témából áll. Ez a mező egyben a kódmező is.
Minden hivatkozást a megjegyzés-file-ba írunk be, ami az idők folyamán terjedelmét tekintve sokkal nagyobb lehetne, mint a törzsfile. A hagyományos file-kezelő rendszerekkel ellentétben azonban csak a ténylegesen szükséges helyet foglaljuk el.
A program listáját a következő oldalakon közöljük. Meglepő lehet az Olvasó számára, hogy a program milyen rövid és mégis minden elvárásnak megfelel. A programhoz azonban még jó pár szubrutin is tartozik a 2. fejezetből, amit helyszűke miatt nem nyomtattunk ki még egyszer. Ha az Olvasó a 2. fejezetben található rutinokat már begépelte és tárolta, itt az alkalom a felhasználásukra.

A programhoz szükséges szubrutinok:

48000	Az adatok formátum szerinti átalakítása
49000	Adatbevitel
50000	Az alapértelmezés szerinti mezők bevitele
51000	A képmaszk felépítése
53000	A megjegyzések olvasása
53200	A következő megjegyzés olvasása
54000	Egy rekord beolvasása
55000	Egy rekord törlése
56000	A megjegyzések bevitele
56600	A következő rekord bevitele
57000	A törzsrekordok bevitele
58000	A file megnyitása
59000	A file előkészítése

A program listája:

```
P33

1000 :
1010 REM *** PROGRAM START ***
1020 :
1025 GOSUB 1500:REM A PROGRAM INICIALIZALASA
1030 RESTORE
1040 FORMP=1TO8:REM A FOMENU BEOLVASASA
1050 READM%(MP),M$(MP),MF%(MP),MT%(MP)
1060 NEXTMP
1070 FORMP=1TO5:REM A FEJLECEK BEOLVASASA
1080 READKO$(MP)
1090 NEXTMP
1100 READTH$,S$
1200 REM A FOMENU KIIRASA
1210 PRINTCHR$(147):REM KEPERNYO TORLESE
1220 FORMP=1TO8
1230 F%=M%(MP):F$=M$(MP):FL%=MF%(MP):FF%=MT%(MP)
1240 GOSUB 51000:REM KEPMASZK KIRAJZOLASA
1250 NEXTMP
1260 REM MENUPONT KIVALASZTASA
1270 F%=17:GOSUB49000
1280 IFF$<"1"ORF$>"6"THEN1270
1290 MP=VAL(F$):ONMPGOSUB3000,4000,5000,6000,7000,8000
```

```
1300 GOTO 1210:REM VISSZA AZ ELEJERE
1500 REM PROGRAM INICIALIZALASA
1510 PRINTCHR$(147)
1520 F%=3:F$="IRODALOMJEGYZEK":FL%=8:FF%=0:GOSUB51000
1530 F%=6:F$="A FILE NEVE: ":FL%=8:FF%=0:GOSUB51000
1540 GOSUB 49000:QN$=F$:PRINT:REM A FILE NEVENEK BEOLVASASA
1560 DN$=QN$+".PR":OPEN15,8,15
1570 GOSUB 63500:CLOSE15:REM LETEZIK ILYEN FILE "?"
1580 IF F1$="00"THENGOSUB58000:RETURN:REM A FILE LETEZIK
1590 GOSUB59060:RETURN:REM A FILE FELEPITESE
3000 :
3010 REM ***** A TERMEK BEOLVASASA *****
3020 :
3030 GOSUB40000:REM A KEPMASZK FELEPITESE
3040 F%=5:F$=TH$:FL%=25:FF%=0:GOSUB51000:GOSUB49000:QD$=F$
3045 IFF$=CHR$(137)THEN3080
3050 F%=7:F$=S$:GOSUB51000:GOSUB49000:QX$=F$
3060 IFF$=CHR$(137)THEN3080
3065 GOSUB57000:REM A TEMA FELIRASA A LEMEZRE
3070 GOSUB56000:REM A HELY KIIRASA
3075 IFVAL(F1$)<>0THEN41000:REM HIBA
3080 RETURN
4000 :
4010 REM ***** A HELY BEOLVASASA *****
4020 :
4030 GOSUB 40000:REM A FEJ NYOMTATASA
4040 F%=5:F$=TH$:FL%=25:FF%=0:GOSUB51000
4050 F$=QD$:GOSUB50000
4060 F%=7: F$=S$: FL%=25:GOSUB51000:GOSUB49000:REM A HELY BEOLVASASA
4070 IFF$=CHR$(137)THEN4100
4080 QX$=F$:IFQT$=".EX"THENGOSUB56600:GOTO4090:REM MEGSZAKITAS
4085 GOSUB56000
4090 IFF1$<>"00"THEN41000
4100 RETURN
5000 :
5010 REM ***** EGY TEMA KERESESE *****
5020 :
5030 GOSUB 40000:REM A FEJ NYOMTATASA
5040 F%=5:F$=TH$:FL%=25:FF%=0:GOSUB51000
5050 F%=7: F$=S$:GOSUB51000
5060 F%=5:GOSUB49000:REM A TEMA BEOLVASASA
5070 IFF$=CHR$(137)THEN5440:REM MEGSZAKITAS
5080 QK$=F$:GOSUB54000:REM A TEMA KERESESE
5090 IFQF%>0THEN5400:REM ILYEN TEMA NINCS
5100 F$=QD$:GOSUB50000:REM KIIRAS A MASZKBA
5110 GOSUB53100
5120 IFQF%>0THEN5400
5130 F$=QX$:F%=7:GOSUB50000
5140 F%=24:F$=F7 - KOVETKEZO HELY / F2 - MEGSZAKITAS":FL%=0:FF%=0
5150 GOSUB51000
5160 GETE$:IFE$=CHR$(137)THEN5440
5170 IFE$<>CHR$(136)THEN5160
5180 REM KOVETKEZO HELY OLVASASA
5190 GOSUB53200
5200 IFQF%>0THEN5400
5210 GOTO5130
5400 REM A TEMA NINCS MEG
5410 F%=24:F$="                                                    ":FL%=0:GOSUB51000
5420 F$="A REKORD NEM LETEZIK.":GOSUB51000
5430 GETE$:IFE$=""ORE$<>CHR$(13)THEN5430
5440 RETURN
6000 :
6010 REM ***** EGY TEMA TORLESE *****
6020 :
6030 GOSUB40000:REM FEJLEC NYOMTATAS
6040 F%=5:F$=TH$:FL%=25:FF%=0:GOSUB51000
```

```
6050 F$=QD$:GOSUB50000:REM AZ AKYUALIS TEMA A KEPERNYORE
6060 F%=7:F$="VALOBAN TOROLJEM?  (I/N)":FL%=1:FF%=1:GOSUB51000
6070 GOSUB49000:REM VALASZ
6080 IFF$="N"THEN6110:REM MEGSZAKITAS
6090 IFF$<>"I"THEN6070
6100 GOSUB55000
6110 RETURN
7000 :
7010 REM ***** A TEMA NYOMTATASA *****
7020 :
7030 GOSUB 40000
7040 F%=5:F$=TH$:FL%=25:FF%=0:GOSUB51000
7050 F$=QD$:GOSUB50000:REM AZ AKTUALIS TEMA A KEPERNYORE
7060 F%=7:F$="A NYOMTATO KESZ (I/N)":FL%=1:FF%=1:GOSUB51000
7070 GOSUB49000
7080 IFF$="N"THEN7300:REM MEGSZAKITAS
7090 IFF$<>"I"THEN7070
7100 GOSUB53100:REM AZ ELSO HELY OLVASASA
7110 IFQF%>0THEN7300:REM A REKORD VEGE
7120 OPEN1,4
7130 FI$=QD$:FO$="AAAAAAAAAAAAAAAAAAAAAAA    AAAAAAAAAAAAAAAAAAAAAAA"
7140 GOSUB48000:REM FORMALAS
7150 FI$=QX$:GOSUB48000
7160 PRINT#1,FP$
7170 FO$="                                    AAAAAAAAAAAAAAAAAAAAAAA"
7175 REM A MARADEK HELYEK FORMALASA
7190 GOSUB53200:REM A KOVETKEZO HELY BEOLVASASA
7200 IFQF%>0THEN7300:REM REKORD VEGE
7210 FI$=QX$:GOSUB48000:REM FORMALAS
7220 PRINT#1,FP$:GOTO7190
7300 CLOSE1:RETURN
8000 :
8010 REM ***** PROGRAM VEGE *****
8020 :
8030 GOSUB 58300
8040 END
10000 :
10010 REM STRINGEK
10020 :
10030 DATA0,"FOMENU",0,0
10040 DATA5,"1 : A TEMA BEVITELE",0,0
10050 DATA7,"2 : A HIVATKOZAS BEVITELE",0,0
10060 DATA9,"3 : A TEMA KERESESE",0,0
10070 DATA11,"4 : A TEMA TORLESE",0,0
10080 DATA13,"5 : A TEMA NYOMTATASA",0,0
10090 DATA15,"6 : A PROGRAM VEGE",0,0
10100 DATA17,"KEREM VALASSZON!",1,1
10110 DATA"A TEMA BEVITELE"
10120 DATA"A HELY BEVITELE"
10130 DATA"A TEMA KERESESE"
10140 DATA"A TEMA TORLESE"
10150 DATA"A TEMA NYOMTATASA"
10160 DATA"TEME : ","HELY : "
40000 :
40010 REM ***** FEJNYOMTATAS *****
40020 :
40030 PRINTCHR$(147):REM KEPERNYO TORLESE
40040 F%=0:F$=KO$(MP):FL%=0:FF%=0:GOSUB51000:REM FEJSOR NYOMTATAS
40050 RETURN
41000 :
41010 REM ***** HIBAKEZELES *****
41020 :
41030 F%=24:F$="HIBA A LEMEZEN":FL%=0:FF%=0:GOSUB51000
41040 GETE$:IFE$=""ORE$<>CHR$(13)THEN41040
41050 RETURN

READY.
```

Már a lista alapján kiderül, hogy a főprogram semmi mást nem tesz, mint hívja a szubrutinokat. Ezen kívül csak a főmenüt és a beolvasást kell megszervezni. Egyébként a program szinte kizárólag szubrutin-hívásokból áll. A rövid változólista is ezt tükrözi:

MP	a választott menüpont
TH$	a "téma:" szöveg
S$	a "hivatkozás:" szöveg
M%	az egyes menüpontok sorszáma
M$	a menüpontok
MF%	a beviteli mezők hossza (0)
MT%	a beviteli mezők típusa (0)
KO$	az egyes beviteli maszkok fejléce

A program négy nagy részre tagolódik:

1000–2000 sor	: a főmenü felépítése és az egyes szubrutinok kiválasztása
2000–8000 sor	: az egyes szubrutinok
10000–20000 sor	: a szükséges állandók
40000–60000 sor	: a szubrutinok

Az egyes programrészek szinte maguktól értetődnek. Mégis egy rövid magyarázat:

1000–2000 sor	: A program inicializálása. A file-név beolvasása és az állandó füzérek értékének beállítása.
1200–3000	: A főprogram; kiírja a főmenüt és ellenőrzi a menüpontot.
1500–1590	: A file-név beolvasása. Ha ilyen nevű file nincs a lemezen, akkor új file-t készítünk elő.
3000–4000	: A téma beolvasása, azaz a törzsadatok beolvasása és tárolása
3030	: A fejléc nyomtatása
3040–3050	: A képernyőmaszk felépítése, valamint a téma és a hivatkozás beolvasása.
3060	: A téma tárolása.
3065	: Ha nincs hivatkozás, a tárolást átugorjuk.
3070	: A hivatkozást felírjuk a lemezre a megjegyzés file-ba.
3075	: Hiba esetén hibajelzés
3080	: Vissza a főmenübe
4000–4100	: A hivatkozás bevitele a témához
4030	: A fejléc nyomtatása
4040–4060	: A beviteli maszk felépítése és a hivatkozás beolvasása
4070	: Üres bevitel esetén megszakítás!
4080	: A hivatkozás tárolása a megjegyzés file-ban, ha előzőleg a file-t még nem használtuk.
4090	: Hiba esetén hibajelzés!
4100	: Vissza a főmenübe
5000–5430	: A téma keresése a lemezen és kiírása hivatkozásokkal együtt.
5030	: A fejléc nyomtatása
5040–5060	: A beviteli maszk felépítése és a keresőfogalom beolvasása
5070	: Üres bevitel esetén megszakítás!
5080	: A téma keresése a lemezen

5090	: Eredménytelen keresés esetén a megfelelő jelzés kiírása
5100	: A téma bevitele a maszkba
5110–5130	: Az első hivatkozás keresése és bevitele a maszkba
5140–5170	: Tovább kell keresni?
5180–5210	: A következő hivatkozás
5400–5430	: A hibajelzés kiírása és vissza a főmenübe
6000–6110	: A téma törlése
6030–6090	: Biztonsági lekérdezés
6100	: A téma törlése
7000–7300	: A téma nyomtatása
7040–7090	: Biztonsági lekérdezés
7100	: Az első hivatkozás olvasása
7120	: A nyomtatócsatorna megnyitása
7130–7140	: A kiírás megformálása (az FO$-ban természetesen tetszés szerint változtathatjuk a formátum utasítást)
7160	: A téma és az első hivatkozás kiírása
7170	: Új formátum a fennmaradó hivatkozások számára
7190–7220	: A következő hivatkozás olvasása és kiírása
7300	: A nyomtatócsatorna lezárása és vissza a főmenübe
10000–10100	: A főmenü füzérei. Minden DATA sor a következőket tartalmazza: sorszám, füzér, a beviteli mező hosszúsága, a beviteli mező típusa
10200–10240	: Az egyes szubrutinok fejlécei.

Néhány szót a program kezeléséről.
Indításkor a program kéri a file nevét. Ha ilyen nevű file nincs a lemezen, akkor egy új file-t készít elő. Legyen a törzsrekord, a megjegyzés, a kódpozíció és a kódmező hossza egyaránt 25. Ezek az értékek természetesen tetszőlegesek, az Olvasó bármelyiket megváltoztathatja. Ebben az esetben azonban a nyomtatást is megfelelően meg kell változtatnia. A főmenüből az egyes programrészeket a szokásos módon választhatjuk ki. Minden programrészből az F2 billentyűvel léphetünk ki. Ha "A téma bevitele" programrész egy új téma és a hozzátartozó első hivatkozás bevitelére szolgál, minden további hivatkozást "A hivatkozások bevitele" programrésszel tárolhatunk. A program mindig azt a témát dolgozza fel, amit "A téma bevitele", illetve "A téma keresése" programrészben jelöltünk meg. "A téma törlése", "A hivatkozások bevitele" és "A téma nyomtatása" programrészek tehát mindig erre vonatkoznak. Ebből következik, hogy ha egy témát meg akarunk változtatni, akkor vagy egy új témát kell bevinnünk, vagy előtte a módosítandó témát megkeresnünk. Az általunk készített program ugyan mindent tud, amit a kiválasztott feladathoz tudni kell, de más feladatokra ebben a formában nem alkalmas. Az Olvasó fantáziájára bízzuk, hogy a programot elegánsabbá és sokoldalúbbá tegye. Reméljük, hogy sikerült bebizonyítani, hogy a 2. fejezet szubrutinjainak segítségével milyen egyszerű file-kezelő programot írni.

Néhány ötlet a program bővítéséhez:
Ha a főmenübe egy újabb programpontot kívánunk beilleszteni, írjuk be az 10000-es sortól kezdve a hozzátartozó füzéreket és növeljük meg a határokat az 1040-es és az 1070-es sorokban, valamint iktassuk be az 1290-es sorba a szubrutin ugrási címét.
A programot sok helyen finomíthatjuk. Írhatunk hozzá néhány sort, amely pl. lehetővé teszi a színek megválasztását, az adatbevitel érvénytelenítését, a JOKER használatát stb. Mi csak arra törekedtünk, hogy felkeltsük az Olvasó érdeklődését.
Ki tudja mivé fejlődhet még ez a program az Olvasó kezében!

4.6 Kényelmes címkezelés

1. RÉSZ:

Ezzel a programmal az Olvasó nyilvántarthatja ismerősei lakcímét. A címek rögzítését, keresését és a címke nyomtatását kényelmessé és egyszerűvé teszi.
Indításkor a következő menü jelenik meg a képernyőn:

CÍMKEZELÉS

MENÜ

–1– A CÍMEK BEVITELE
–2– A CÍMEK TÁROLÁSA
–3– A CÍMEK BEOLVASÁSA LEMEZRŐL
–4– NYOMTATÁS
–5– KIÍRÁS A KÉPERNYŐRE
–6– VÉGE

KÉREM VÁLASSZON!

Az egyes menüpontok:

–1– A CÍMEK BEVITELE

Ebben a programrészben bevihetjük a nyilvántartásra szánt címeket. Az előre meghatározott beviteli forma a következő:

MEGSZÓLÍTÁS: (ÚR, HÖLGY, CÉG stb.)
NÉV:
CÍM: (UTCA VAGY POSTAFIÓK)
IRÁNYÍTÓSZÁM/HELYSÉG:

A cím bevitele után a program megkérdezi, hogy tárolja-e az adatokat, folytassa-e a beolvasást, vagy visszatérjen a menübe. A kérdésre a megfelelő funkcióbillentyűvel válaszolhatunk.

–2– A CÍMEK TÁROLÁSA
–3– A CÍMEK BEOLVASÁSA LEMEZRŐL

A 2. és 3. menüpont feladatát a nevük egyértelműen elárulja.

–4– NYOMTATÁS

A címek nyomtatására két lehetőség kínálkozik:
Válogatás: Ha erre a kérdésre "I"-vel (igennel) válaszolunk, a négy adatmező (megszólítás/

név/cím/postai irányítószám-helység) egyike szerint kikeresett címet nyomtathatjuk ki.
Ha a kérdésre N-nel (nemmel) válaszolunk, akkor a nyomtató a tárolt címeket egymás után kinyomtatja. A formátum egy 33 oszlopból és 9 sorból álló etikettnek (cimkepapír) felel meg.

–5– KIÍRÁS A KÉPERNYŐRE

Ez a programrész elvileg ugyanúgy dolgozik, mint a –4– pontban ismertetett programrész, azzal az eltéréssel, hogy a kiírás a képernyőre történik.
A továbblapozáshoz minden cím után le kell nyomnunk az F1 billentyűt. A file végének elérésekor az ADATOK VÉGE jelzés jelenik meg, ekkor az F3 billentyű megnyomásával visszatérhetünk a menübe.

–6– VÉGE

A 6-os menüpont a munkát befejezettnek tekintve megszakítja a program futását.

2. RÉSZ:

A program központi része a menü. Innen vezéreljük az egyes programfunkciókat és minden művelet után ide térünk vissza.
A program első részében felépítjük a menüt, mégpedig úgy, hogy az egyes tevékenységeket számokkal jelöljük meg, ami megkönnyíti az elágazást. A kívánt funkció lekérdezésére a GET utasítást használjuk. Ennek két oka van:
a) A program „gombnyomásra" folytatódik, azaz a gépkezelőnek egy billentyűt kell leütnie.
b) Mivel a VAL függvény a füzért számmá alakítja át, könnyebbé válik a hiba lekérdezése. Ha a kapott érték a megfelelő határokon belül van, akkor az ON..GOSUB utasítással kényelmesen megoldhatjuk az elágazásokat.
A 130-as sorra – ugrás az 5-ös sorra – azért van szükség, hogy a rutinokból való visszatérésnél újra felépíthessük a maszkot.
Még egy kis útmutatás azoknak, akik az adatmezőket igényeikhez kívánják igazítani. A rekordok maximális számát egy változó határozza meg (lásd változólista), értéke a programban 200.
A címkepapír formátuma 33 oszlop/9 sor. Ha ezen változtatni akarunk, az oszlopszám változóját és a 4110–4199 sorokban található kifejezést kell módosítani.

A program változói:

MAX	A rekordok maximális száma
SP	Az etikettek oszlopszáma
AD$(**)	A címeket tartalmazó kétindexes változó
W$	A választást tartalmazó karakter
W	Az elágazást vezérlő változó
A	A rekord sorszáma
I,J	Ciklusváltozók
S1	A keresési szempontok sorszáma
S1$	A keresett mező
1S	A keresett mező hossza

P34

```
0 REM *** CIMEK V. 1.1 DECEMBER 83 ***
1 MAX=200 : SP=33
2 DIMAD$(MAX,4)
5 PRINT CHR$(147)+CHR$(142)+CHR$(8)
10 REM ** A KEPERNYO INICIALIZALASA :*
20 POKE 53280,2 : POKE53281,2 : PRINT CHR$(144)
22 REM ** A MENU FELEPITESE **
23 PRINT SPC(10);CHR$(18)"CIMNYILVANTARTAS":PRINT
29 PRINT"****************************************";
30 PRINT"*"SPC(38)"*"; : PRINT"*"SPC(16)"MENU"SPC(18)"*"; : PRINT"*"SPC(38)"*";
31 PRINT"*       -1-A CIMEK BEVITELE              *"; : PRINT"*"SPC(38)"*";
32 PRINT"*       -2-A CIMEK TAROLASA              *"; : PRINT"*"SPC(38)"*";
33 PRINT"*       -3-A CIMEK BEOLVASASA            *"; : PRINT"*"SPC(38)"*";
34 PRINT"*       -4-NYOMTATAS                     *"; : PRINT"*"SPC(38)"*";
35 PRINT"*       -5-KIIRAS A KEPERNYORE           *"; : PRINT"*"SPC(38)"*";
36 PRINT"*       -6-VEGE                          *"; : PRINT"*"SPC(38)"*";
37 PRINT"****************************************";
38 PRINT"*    KEREM VALASSZON!                    *";
39 PRINT"****************************************";
100 GETW$:IFW$=""THEN100
110 W=VAL(W$):IFW<1ORW>6THEN100
120 ONWGOSUB1000,2000,3000,4000,5000,9999
130 GOTO5
1000 A=A+1
1010 PRINT CHR$(147);CHR$(18)"CIMEK BEOLVASASA";A:PRINT:PRINT:
1020 INPUT"MEGSZOLITAS:";AD$(A,1) : PRINT
1021 IF LEN(AD$(A,1))> SP THEN 1020
1022 INPUT"NEV    : ";AD$(A,2) : PRINT
1023 IF LEN (AD$(A,2)) >SP THEN 1022
1024 INPUT"CIM: ";AD$(A,3) : PRINT
1025 IF LEN(AD$(A,3)) >SP THEN 1024
1026 INPUT"IRANYITOSZAM/HELYSEG: ";AD$(A,4) : PRINT
1027 IF LEN(AD$(A,4)) >SP THEN 1026
1028 PRINT : PRINT CHR$(18)"F1=TOVABB"SPC(3)"F3=TAROLAS"SPC(3)"F5=MENU"
1030 GETA$
1040 IFA$=CHR$(133)THEN1000
1050 IFA$=CHR$(134)THENGOSUB2000:RETURN
1060 IFA$=CHR$(135)THENRETURN
1070 GOTO1030
2000 PRINT CHR$(147);CHR$(18)"CIM A LEMEZRE":PRINT:PRINT
2010 OPEN1,8,15,"S:CIMEK.DAT":CLOSE1
2020 OPEN1,8,2,"CIMEK.DAT,S,W":PRINT#1,A
2030 FORI=1TOA:FORJ=1TO4
2040 PRINT "AZ "I". SORSZAMU CIMET IROM";CHR$(145)
2050 PRINT#1,AD$(I,J):NEXT:NEXT:CLOSE1
2060 RETURN
3000 PRINT CHR$(147);CHR$(18)"CIMEK":PRINT:PRINT
3020 OPEN1,8,2,"CIMEK.DAT,S,R":INPUT#1,A
3030 FORI=1TOA:FORJ=1TO4
3040 PRINT"AZ"I". SORSZAMU CIMET OLVASOM";CHR$(145)
3050 INPUT#1,AD$(I,J):NEXT:NEXT:CLOSE1
3060 RETURN
4000 PRINT CHR$(147);CHR$(18)"NYOMTATAS":PRINT:PRINT
4010 OPEN4,4,7
4020 INPUT"VALASZTAS J/N";EG$ : IFEG$="J"THENGOSUB4500 : RETURN
4030 FORI=1TOA : GOSUB4110 : NEXT : CLOSE4 : RETURN
4100 REM ** ETIKETTNYOMTATAS **
4105 PRINT"AZ"I". CIMET NYOMTATOM";CHR$(145)
4110 PRINT#4
4120 PRINT#4,AD$(I,1)
4130 PRINT#4
4140 PRINT#4,AD$(I,2)
4150 PRINT#4
4160 PRINT#4,AD$(I,3)
```

```
4170 PRINT#4,PRINT#4,AD$(I,4)
4190 PRINT#4
4199 RETURN
4500 PRINT CHR$(147);CHR$(18)"VALASZTAS" : PRINT : PRINT
4510 PRINT"KERESESI SZEMPONT:"
4511 PRINT"-1-MEGSZOLITAS"
4512 PRINT"-2-NEV"
4513 PRINT"-3-CIM"
4514 PRINT"-4-IRANYITOSZAM/HELYSEG" : PRINT
4515 PRINT"AZ ON VALASZTASA?";
4520 GETW$ : IFW$=""THEN4520
4530 W=VAL(W$) : IFW<1ORW>4THEN4520
4540 PRINTW : SL=W
4550 PRINT : PRINT"A KERESETT ADAT:" :INPUTSL$ : LS=LEN(SL$)
4600 FORI=1TOA
4610 IF LEFT$(AD$(I,SL),LS)=SL$ THENGOSUB4100
4620 NEXT : CLOSE4 :RETIURN
5000 PRINT CHR$(147);CHR$(18)"KIIRAS A KEPERNYORE":PRINT:PRINT
5020 INPUT"KERESES I/N";EG$ : IFEG$="J"THENGOSUB5500 : RETURN
5030 FOR I=1TOA : GOSUB 5100 : NEXT
5040 PRINT : PRINT"NINCS TOBB ADAT!!" : PRINT CHR$(18)"=> F3"
5050 GET F3$ : IFF3$<>CHR$(134)THEN5050
5060 RETURN
5100 PRINT CHR$(147);CHR$(18)"AZ"I". CIM": PRINT : PRINT
5110 FOR J=1TO4
5120 PRINT AD$(I,J) : NEXT
5130 PRINT : PRINT CHR$(18)"TOVABB F1-EL"
5140 GET F1$ : IF F1$<>CHR$(133)THEN5140
5150 RETURN
5500 PRINT CHR$(147);CHR$(18)"KERESES" : PRINT : PRINT
5510 PRINT"KERESESI SZEMPONTOK:"
5511 PRINT"-1-MEGSZOLITAS"
5512 PRINT"-2-NEV"
5513 PRINT"-3-CIM"
5514 PRINT"-4-IRANYITOSZAM/HELYSEG" : PRINT
5515 PRINT"KEREM VALASSZON!";
5520 GETW$ : IFW$=""THEN5520
5530 W=VAL(W$) : IFW<1ORW>4THEN5520
5540 PRINTW : SL=W
5550 PRINT : PRINT"A KERESETT ADAT" : INPUTSL$ : LS=LEN(SL$)
5600 FORI=1TOA
5610 IF LEFT$(AD$(I,SL),LS)=SL$ THENGOSUB5100
5620 NEXT : PRINT : PRINT"NINCS TOBB ADAT" : PRINT CHR$(18)"=>F3"
5630 GET F3$ : IFF3$<>CHR$(134)THEN5630
5640 RETURN
9999 PRINT CHR$(147):PRINTTAB(18)"VEGE":CLR:END

READY.
```

5. FEJEZET
A PROGRAMOZÁSI SEGÉDLETEK FELHASZNÁLÁSA

5.1. A MASTER 64

A programozó fegyvertára a szubrutin könyvtár.
A 4. fejezetben ismertetett programokat szinte kivétel nélkül az előre elkészített szubrutinokból építettük fel. Reméljük sikerült bebizonyítanunk, hogy a programozói munkát hihetetlenül megkönnyíti egy jó programkönyvtár. Bizonyára az Olvasó is bővítgeti a sajátját a különböző számítógépes folyóiratokra és szakkönyvekre, és persze nem utolsósorban a saját ötleteire támaszkodva. Míg a kedvtelésből programozók számára az új rutinok és programtrükkök kidolgozása gyakran öncélú és elsősorban szórakozás, addig a hivatásos programozó a gyűjtögetés során figyelmét főként a programok használhatósága felé fordítja. Az a sok rutin, amire neki a hatékony programkészítéshez szüksége van, csupán eszköz számára, és mi sem egyszerűbb, mint az ilyen eszközöket egyszerűen megvásárolni. Ez ugyan nem okoz annyi örömöt, mint a saját fejlesztés, de sok időt és végül is pénzt takarít meg.
A Commodore 64-eshez már egész sor olyan BASIC bővítést és segédeszközt kínálnak, amelyeknek az a feladata, hogy megkönnyítsék és kényelmessé tegyék a programkészítést. A sokféle segédszoftver közül kiválasztottunk egyet, a MASTER 64-et, amelyet a legalkalmasabbnak tartunk arra, hogy ismertetésével meggyőzzük az olvasót az ilyen segédeszközök hasznosságáról.
A MASTER 64 egy nagyon alaposan kidolgozott átfogó rendszer, amely megérdemli a „programfejlesztő rendszer" elnevezést.
A MASTER-t eredetileg a nagyszámítógépeken használt programozó eszközök mintájára a 8000-es sorozat nagy Commodore számítógépeire fejlesztették ki. Napjainkban már sok változat létezik, a 8032-es, a 8096-os, a 64-es, a 600-as és a 700-as Commodore számítógépekhez. Minden változat a legmesszebbmenőkig azonos, így a MASTER segítségével készített programok könnyen átvihetők más Commodore számítógépekre. A MASTER 64-ben érvényesülő alapgondolatok nagyrészt azonosak az általunk képviselt és a könyvben ismertetett programozási alapelvekkel és a rendszer tartalmazza mindazokat az elemeket, amelyek igényes programok egyszerű elkészítéséhez szükségesek, vagyis:

- kényelmes képernyőkezelés a képernyőmaszk gyors készítéséhez
- ISAM file-kezelés a gyors adathozzáféréshez és a hatékony file-kezeléshez
- nyomtató generátor tetszőleges kivitelű maszkok egyszerű készítéséhez és teszteléséhez
- programvédelem "nolist" móddal és egyéni kódokkal
- többszörösen pontos aritmetika, minden alapműveletre 22 helyiérték pontossággal
- BASIC bővítések: TOOLKIT funkciók és a teljes BASIC 4.0 változat.

Ahhoz, hogy az Olvasó betekintést nyerjen a MASTER 64 programcsomag felhasználási lehetőségeibe, ismertetjük néhány utasítását.

A MASTER képernyőmaszk generátor

A könyvbeli adatbeviteli szubrutinok elemzése közben bizonyára kialakult az Olvasóban egy benyomás arról, hogy mekkora ráfordítás szükséges egy kényelmes, biztonságos, univerzális és emellett még gyors beviteli rutinhoz. A MASTER a miénkhez hasonló koncepcióval, szintén

a képmaszkokra építi az adatbevitelt, ehhez kész BASIC utasításokat bocsát a programozó rendelkezésére.
A képmaszk zónákból épül fel. A képernyőzóna nem más, mint egy rögzített hosszúságú mező a képernyő adott helyén (sor, oszlop). A MASTER segítségével maximum 127 olyan zónát definiálhatunk, amire azután be- és kiviteli utasítások vonatkoznak.

Az új utasítások célszerű sorrendje:

1. A zóna meghatározása a képernyőn elfoglalt pozíció (sor, oszlop), a zónahosszúság megadásával, valamint egy esetleges formátum-előírás a numerikus adatok automatikus formátumozásához.
2. Adatkérés egy zónából. A zónában megjelenik a kurzor, ami jelzi, hogy begépelhetjük az adatokat. Adatbevitel közben a kurzorbillentyűt tetszés szerint használhatjuk, a MASTER megengedi, hogy a zónából kilépjünk és ezzel a képmaszkot elrontsuk. Az adatok átvétele RETURN-nel vagy egyéb, a programozó által meghatározandó ellenőrző billentyűkkel történik.
3. A zónatartalom átvétele egy változóba. Egy MASTER utasítással egy zóna tartalmát (a zónaszám határozza meg) egy BASIC változóban tároljuk.
4. Kiírás egy zónába. Egy változót vagy egy tetszőleges kifejezést formátumozva kiírhatunk a képernyő valamely zónájába.

A 2...4. lépést tetszőleges gyakorisággal használhatjuk egy programban, de ezeket definiálnunk csak egyszer kell.
A kényelem további fokozására további utasítások állnak a programozó rendelkezésére: vonalak húzásához, adatok tetszőleges képernyőpozíciókba történő írásához, törléshez, invertáláshoz és a képernyő tetszőleges részeinek pörgetéséhez (scroll) stb.
Ha egy programban több képernyőmaszkkal kívánunk dolgozni, a MASTER segítségével olyan virtuális képernyőket definiálhatunk az operatív tárban, amelyeket feltölthetünk adatokkal, miközben a program egy másik képmaszkkal dolgozik.
A virtuális képernyőt a későbbiekben felcserélhetjük a valódi képernyővel. Különösen hasznos az a lehetőség, hogy a teljes képernyő-oldalakat, beleértve a zónadefiníciókat is, a lemezen tárolhatjuk és onnan bármikor visszatölthetjük. Ezzel sok helyet takarítunk meg a tárban, hiszen a képernyőmaszk felépítése egy külön programmal történhet, amit csak egyszer kell lefuttatni. Ugyanakkor a képernyőmaszkot is megváltoztathatjuk anélkül, hogy a tulajdonképpeni program bármiféle módosításra szorulna.

A képernyőkezelés utasításai:

DECZ, N, Z, S, L, (,TY) (,F$)	Az L hosszúságú N számú zónát deklarálja Z sortól, S oszloptól. Kiegészítőleg még megadhatunk egy típust (,TY) ami azt jelenti, hogy ez a zóna például csak numerikus adatokat fogad el. Az adatok formátumát az F$ füzérben adhatjuk meg.
REQZ N	Adatbevitel az N zónába
INZ N,A$	A beolvasott adat tárolása egy változóban
OUTZ N,A$	Adatok kiírása egy zónába
CLEARZ N	Egy zóna tartalmának törlése
REVZ N	A képernyőn egy zónát invertál
OUT A$,Z,S	Z sortól, S oszloptól kezdve kiírja az A$ füzért

CLEAR L,Z,S	A képernyőn egy mezőt töröl
REV Z,S,LZ,LS	Egy LZ hosszúságú és LS mélységű képernyőmezőt invertál
SCROLL Z,S,LZ,LS,TY	Egy képernyőmezőt görget fel vagy le
TLINE L,Z,S	Egy L hosszúságú vízszintes vonalat húz S sortól, S oszloptól
TCOL L,Z,S	Egy L hosszúságú függőleges vonalat húz Z sortól, S oszloptól
SSAVE U,FN$	FN$ néven egy képernyőmaszkot tárol a lemezen
SLOAD U,FN$	Egy képernyőmaszkot tölt be a lemezről
SSET N	Egy virtuális képernyőt szelektál
SCLEAR	A szelektált képernyőt tölti be
SCOPY	Egy virtuális képernyőt a valódi képernyőre másol
SEXCH	Két képernyő tartalmát kicseréli.

A fenti utasításokból összeállítottuk egy adatbevitel programját.

```
100 DECZ 1,5,20,6,N: REM 1. zóna, numerikus
110 DECZ 2,7,15,25 : REM 2. zóna, alfanumerikus
120 OUT "Vevőszám",5,1 : REM 5. sor, 1. oszlop
130 OUT "Név",7,1
140 REQZ 1 : REM 1. zóna bevitele
150 INZ 1,KN : REM vevőszám beolvasása
160 REQZ 2 : REM 2. zóna bevitele
170 INZ 2,NM$ : REM név beolvasása
180 GOSUB 1000 : REM adatok feldolgozása
190 CLEARZ 1,2 : REM 1+2. zóna törlése
200 GOTO 140 : REM következő bevitel
```

A MASTER nyomtatómaszk generátor

A képernyőmaszk generátornál már megismert lehetőségeket nyomtatásra is használhatjuk. Először meg kell adnunk a nyomtatott lap felső, alsó és bal szegélyét. A definíciót tároljuk lemezen. Ezután a képernyőhöz hasonlóan definiálhatjuk a nyomtatási zónákat, melyeket ismét a sor, az oszlop, a hosszúság és a formátum határoz meg.
A nyomtatásképbe állandó szövegeket is beilleszthetünk, például nyomtatványokhoz vagy feliratokhoz. A szövegeket programon kívül is megadhatjuk. A nyomtatómaszkot csak a program által előállított adatokkal töltjük ki, a kiírás nemcsak soronként, hanem tetszőleges sorrendben is történhet. A kitöltött maszkot többször is kinyomtathatjuk, ha szükséges (teszteléshez a képernyőre is) kiírathatjuk.
Ha a nyomtatási mezőket töröljük, a program ezeket újra kitöltheti. Összességében mindazokat a lehetőségeket használhatjuk, mint a képernyőmaszkoknál (az adatbevitel kivételével).

A nyomtatás utasításai:

PCREATE U,FN$,Z,S,RO,RU,RL	Egy Z sorú és S oszlopú RO felső szélű, RU alsó szélű, és RL bal szélű nyomtatott oldalt definiálunk
POPEN	Egy nyomtatott oldal megnyitása a feldolgozáshoz
PDECZ N,Z,S,L(,TY) (,F$)	Egy nyomtatási zónát definiálunk (egy képernyő-zónához hasonlóan)

POUTZ N,A$	Kiírás egy nyomatási zónába
PCLEARZ N	Egy nyomtatási zóna tartalmának törlése
PPRINT LF	Egy oldal nyomtatása
PCLEAR L,Z,S	Adatmező törlése
POUT A$,Z,S	Z sortól, S oszloptól kezdve kinyomtatja a füzért
PCOLSE	Egy oldal lezárása
PERASE	Egy oldal törlése

A MASTER-ISAM file-kezelés

A file-kezelés terén a MASTER a legkényelmesebb lehetőséget kínálja, az ISAM file-t. Az index-soros hozzáférési módszernél minden rekordot egy úgynevezett hozzáférési kóddal, az indexszel érünk el. A címfile-ban ez például a vezetéknév lehet. A gépkezelőnek sem a rekord file-on belüli pozícióját, sem a rekordszámot nem kell megjegyeznie – a keresés egyszerűen a névvel történik. Az ISAM file-t menüvezérléssel, egy file-foglaló programmal állíthatjuk össze, ami a file minden paraméterét lekérdezi: a rekord hosszúságát (maximum 254 karakterig), a kód hosszúságát (30 karakterig) és a rekordok maximális számát.
A file méretét a rendszer nem korlátozza, a MASTER-file a teljes lemezterületet elfoglalhatja.

A MASTER file-kezelő utasításai:

IOPEN LF,U,FN$	Egy LF logikai file-számú MASTER-ISAM file-t nyit meg FN$ néven az U egységszámú periférián (az egységszám általában 8)
IWRITE LT, RC$	Felír egy új rekordot a file-ba (R$)
IREAD LF,KY$,RC$	a KY$ hozzáférési kódhoz tartozó rekordot beolvassa az RC$ változóba.
IEXIST LF,KY$	Azt vizsgálja, hogy a KY$ hozzáférési kódhoz létezik-e rekord
IUPDATE LF, RC$	Felír egy módosított rekordot (RC$) a file-ba
IDELETE LF,KY$	Egy rekordot töröl
ISTART– LF,KY$,MK$,RC$	Az olvasást növekvő sorrendben indítja. Az MK$ kiválasztó maszk, ha csak bizonyos rekordot keresünk.
ISTART LF,KY$,MK$,RC$	Mint fent, de csökkenő sorrendben
INEXT– LF,MK$,RC$	Beolvassa a következő rekordot
INEXT LF,MK$,RC$	Mint fent, de csökkenő sorrendben
IADD LF, RC$	Felír egy rekordot a hozzáférési kód automatikus beillesztése nélkül a file-ba
IVALIDATE LF	Az IADD-vel írt rekordok indexének beillesztése
IRESTORE LF	Pozicionálás az először írt rekordra
IDATA LF,MK$,RC$	A rekordok olvasása beviteli sorrendben
IUPGRADE LF	Az index biztosítása a lemezen
ICLOSE LF	Egy MASTER-ISAM file lezárása.

A rendszer minden file- művelet után aktualizálja az állapotváltozó értékét, amely a művelet megfelelő végrehajtásáról informálja a programozót. A MASTER tömörítve tárolja a rekordokat, azaz minden felesleges (redundáns) információt eltávolít, körülbelül 10–50%-os helymegtakarítást eredményezve.

A maximális rekordhossz kb. 400 karakter.
A MASTER az adatok biztonságos tárolását sem hanyagolja el: ha a lemezen egy index megsérül, behívhatunk egy programot, amely a teljes file-t rekonstruálja.
A következő program bekér egy vevőnevet és az ISAM file-ban megkeresi a megfelelő rekordot:

```
100 IOPEN 1,8 "O:VEVŐ"
110 REQZ L : INZL, KN$ : REM vevőnév
120 IREAD L, KN$ DA$ : REM a rekord olvasása
130 OUT DA$, 10,1 : REM a rekord kiírása a képernyőre
140 CLEARZ L : REM a beviteli mező törlése
150 GOTO 110
```

A MASTER BASIC

A MASTER a fentieken kívül a következő BASIC-bővítésekkel rendelkezik: Többszörösen pontos aritmetika, minden alapművelet 22 helyiértékkel végezhető. Van továbbá néhány olyan utasítás, amely leegyszerűsíti a lemezhez való közvetlen hozzáférést:

UPLOW	Nagybetűk és kisbetűk felcserélése
DATEPACK	A dátum tömörítése és érvényességének ellenőrzése
DATEUNPACK	A dátum visszaállítása
GOTO ZN	Kiszámított GOTO utasítás, amellyel helyettesíthetjük az ON..-GOTO sorokat.
GOSUB ZN	Kiszámított GOSUB
MADD N1$,N2$,N3$	Összeadás 22 helyiértékkel
MSUB N1$,N2$,N3$	Kivonás 22 helyiértékkel
MMUL N1$,N2$,N3$	Szorzás 22 helyiértékkel
MDIV N1$,N2$,N3$	Osztás 22 helyiértékkel
CREATST A$,L(,A)	Előállít egy L hosszúságú füzért az "A" ASCII kóddal megadott karakterekből
PUTST A$;TY,B$,P,L	Az adatok tömörítése
GETST A$;TY,B$,P,L	Az adatok visszaállítása
INBLOCK TR,SE(,D) (,U)	Beolvassa a lemez adott sávjának (TR) adott szektorát (SE) egy pufferbe
PUTBLOCK TR,SE (,D) (,U)	Felír egy adatblokkot egy pufferből a lemezre
TAKBU A$,P,L	Átvitel egy pufferből egy füzérbe
PUTBUF A$,P,L	Átvitel egy füzérből egy pufferbe
HUNT CHR$(A),A$,P	Megkeresi A$-ban az "A" kódú karaktert
NOLIST A$,D(D)	Tárolás NOLIST módban, a programot védi listázás ellen
ONERROR ZN	Lemezhibák kezelése.

Néhány megjegyzés az adatok tömörítéséhez:

A MASTER alfanumerikus füzér, lebegőpontos és fixpontos számok tömörítésére képes. A tömörítést a PUTST utasítás, a visszaállítást pedig a GETST utasítás végzi el. A típustól függően maximum kb. 50%-os helymegtakarítást érhetünk el.

A fentieken kívül a MASTER tartalmaz egy nagyon hasznos programot, a TOOLKIT programot, amit ebben a könyvben gyakran emlegettünk. A TOOLKIT utasításai:
AUTÓ, DELETE, RENUMBER, DUMP, IF..THEN..ELSE, HARDCOPY, TRACE ON..OFF. A TOOLKIT utasítások megkönnyítik a programok bevitelét, szerkesztését és tesztelését.
Az ISAM file-kezelés még számos segédfunkciót is nyújt, mint a soros file-ok átalakítása ISAM-file-okká és fordítva, az ISAM file-ok Dump funkciói stb.
A MASTER-rel írt programok minden MASTER-rel rendelkező Commodore számítógépen használhatók. Hogy a 64-es programokat más számítógépen is használhassuk, a MASTER 64 tartalmazza a teljes BASIC 4.0 változatot olyan kényelmes lemezkezelő utasításokkal, mint például a DLOAD a DSAVE, a SCRATCH, a RECORD stb.
A MASTER 64 szoftverrel minden sokkal egyszerűbb!

5.2 A STRUKTO 64

Az 1. fejezetben bemutattuk a strukturált programozás alapelveit. Azt is láttuk, hogy többek állítása ellenére BASIC nyelven is lehet strukturált programokat készíteni, bár a BASIC nyelv nem támogatja kifejezetten ezt a programozási módszert.
Hogyan lehetne mégis egyesíteni a BASIC nyelv kényelmes szolgáltatásait olyan szintű strukturálhatósággal, mint amit egy magasabb szintű programnyelv pl. a PASCAL nyújt?
A megoldás a STRUKTO 64 szoftver!
A STRUKTO 64 egy strukturált programozási nyelv, amellyel az 1.6 alfejezetben ismertetett Nassi-Shneidermann diagramokat közvetlenül programszöveggé alakítunk át. A kapott programok ugyanolyan világosak és áttekinthetőek, mint a struktogramok. A STRUKTO 64 rendszer nyelve pontosan olyan egyszerű, mint a BASIC, és ráadásul bele van építve egy TOOLKIT segédprogram is. A programsorokat a BASIC-hez hasonlóan itt is sorszámokkal látjuk el, a sorszámok azonban nem a GOSUB és GOTO utasítások ugrási címei, hanem egyszerű azonosítók a listázáshoz.
Ennyit bevezetőként, és most lássuk a rendszer vezérlő struktúráit:

1. A feldolgozó utasítások

Ezek pontosan olyan utasítások, mint amelyek a BASIC-ből ismertek, és amelyeket az interpreter azonnal végrehajt. Például:

```
140 PRINT A$ ;"DATUM:" ;DT$
```

2. Az alprogramhívás

A STRUKTO 64-ben kétféle szubrutin létezik; sorszámmal, ill. névvel hívható szubrutin. Sorszámmal ellátott szubrutin pl. a következő:

```
3000 BEGIN 6
3010 INPUT "6.ADAT:";A$(6)
3020 PRINT
3030 END
```

Névvel ellátott szubrutin:

```
4000 ENTRY RAJZOLÁS
4010 X=COS(T)*80+90
4020 Y=SIN (T)*80+90
4030 LINE 0,0;X,Y
4040 EXIT
```

A sorszámmal ellátott szubrutint a megszokott GOSUB 6, a névvel ellátott szubrutint pedig a PASS RAJZOLÁS utasítással lehet meghívni.

3. A kétszeres elágazások

A kétszeres elágazásokat a STRUKTO 64-ben is az IF utasítás vezérli. Pl.:

```
100 IF A=0
110 PRINT "A EGYENLO 0"
120 IFNOT
130 PRINT "A NEM EGYENLO 0"
140 IFEND
```

Az IF és az IFNOT utasítások végrehajtására csak akkor kerül sor, ha a feltétel teljesül, egyébként a vezérlés az IFNOT – IFEND közötti programrészre kerül. Ha az IFNOT rész hiányzik, és a feltétel nem teljesül, végrehajtás során a teljes programrész kimarad.
A szálak azonban minden esetben az IFEND utasítást követő utasításon futnak össze ismét.

4. A többszörös elágazások

A többszörös elágazások szervezésére a STRUKTO 64 külön utasítással szolgál. Ez a CASE utasítás. A CASE utasításszó mögött tetszőleges kifejezés állhat, majd néhány programrész, amelyeket a kifejezés értékétől függően alternatív módon kell végrehajtani. A programágak a CASEEND utasítás mögött találkoznak. Pl.:

```
150 INPUT"KEREK EGY 0 ÉS 10 KÖZÉ ESŐ SZAMOT:"; A
160 CASE A
170 OF 0,2,4,6,8,10
180 PRINT "A SZAM PAROS"
190 OF 1,3,5,7,9
200 PRINT "A SZAM PARATLAN"
210 OFREST
220 PRINT "ERVENYTELEN SZAM"
230 CASEEND
```

5. A ciklus

A STRUKTO 64-ben különböző típusú ciklusokkal dolgozhatunk:

a) Kezdőfeltétellel ellátott ciklus

```
100 WHILE FELTETEL
110 :
120 :
130 :
140 WHILEEND
```

A WHILE és a WHILEEND utasítások közötti programrész csak a feltétel teljesülése esetén hajtódik végre.

b) Megszakítási feltétellel lezárt ciklus. Pl.:

```
100 REPEAT
110 INPUT "IRJ BE 1-ET VAGY 0-T";X
120 UNTIL X=0 OR X=1
```

c) Megszakítási feltétel a ciklus belsejében

Ebben a típusban a ciklus elejét a LOOP, a ciklus végét a LOOPEND utasítás jelzi. A két fenti utasítás között bárhol elhelyezhetünk egy megszakítási feltételt, amelynek formátuma: ON feltétel LEAVE. Példa:

```
400 LOOP
410 ....
420.....
430 ON feltétel 1 LEAVE
440 ....
450 ....
460 ON feltétel 2 LEAVE
470 LOOPEND
```

d) Ciklusváltozóval szervezett ciklus

Ez a típus megfelel a BASIC nyelvből ismert ciklusutasításnak. Kezdetét a FOR, végét a FOREND utasítás jelzi.
A fenti vezérlőutasításokon kívül a STRUKTO 64 tartalmaz még egy sor bővítést, amely megkönnyíti a programozást. Lássunk ebből is némi ízelítőt:

A TOOLKIT

A programok begépelését, javítását, tesztelését, tárolását és betöltését megkönnyítő utasítások:

AUTO	Automatikus sorszámozás
REN	Átsorszámozás
INDENT	Vezérlőstruktúrák kiiktatása
TRACE	Nyomkövetés. A program végrehajtása közben kiírja az aktuális sorszámot.
&	Kikapcsolja a TRACE, ill. AUTO üzemmódot.
FIND	Keresés a programszövegben
CONT	A STOP utasítással megszakított program végrehajtását folytatja.
NEW	Törli a tárból a programot.
DEL	Töröl egy meghatározott programrészt.
OLD	Visszaállítja a törölt programot.
EDIT	Kiírja a javítandó programsort.

A funkcióbillentyűket azonosíthatjuk egy karaktersorozat (maximum 10 karakter) begépelésével, így ezek a billentyűk több billentyűt helyettesíthetnek egyszerre.

A lemezkezelő utasítások:

APPEND	A lemezről betöltött programot a tárbeli program alá illeszti.
DIR	Kiírja a képernyőre a lemez tartalomjegyzékét.
!	Beolvassa és kiírja a lemezegység parancscsatornáját.
DISK	Utasítást küld a lemezegységhez.
MON	A külső egységről érkező, vagy külső egységhez továbbított adatokat kiírja a képernyőre.

A LOAD, SAVE, VERIFY utasításokban nem kell minden esetben megadni az egységszámot. A feldolgozás elején beállított egységszám mindaddig érvényben marad, amíg újat nem adunk. A pillanatnyi egységszámot a rendszer a VN% változóban tárolja.
A DISK utasítások továbbítják a parancsot az egységhez, és helyettesítik az alábbi típusú BASIC utasításokat:

OPEN 15,8,15,"...":CLOSE 15

Ezáltal a gép folytathatja a feldolgozást, miközben a lemezegység dolgozik.

A STRUKTO 64 egy további sajátossága, hogy segítségével minden probléma nélkül összefűzhetjük a BASIC programokat. Tegyük fel, hogy több különálló programot megírtunk, és ezeket egy programmá szeretnénk összefűzni, amelyben az eredetileg megírt részek egy-egy szubrutin szerepét töltik be. Az összeillesztést a STRUKTO 64-gyel a következőképpen végezhetjük el:
Minden program elejére beírjuk az ENTRY név, végére pedig az EXIT utasítást és ezeket tároljuk a lemezen. Ha készen vagyunk, töröljük a tárat és megírjuk a főprogramot, amelyből az egyes részeket hívni fogjuk.
Végül az APPEND utasítással sorban a főprogram alá töltjük a lemezen tárolt programrészeket.
Ebben semmi különös nincs, ezt egy BASIC programmal is feltehetően meg tudjuk csinálni. Tegyük fel azonban, hogy az egyik programegység DATA sorokat tartalmaz. Mivel ezeket a DATA sorokat BASIC programban általában a program elején kell beolvasni, ez lehet, hogy zavart okoz. A STRUKTO 64-ben ez a probléma nem létezik. Ugyanis minden szubrutinban elhelyezhetünk olyan DATA sorokat, amelyeket csak az a szubrutin olvashat be, és így a szubrutinban elhelyezkedő DATA sorok nem zavarják a főprogram által megszabott beolvasási sorrendet.
Ha az egyes programrészek túlságosan hosszúak, és nem férnek el egyszerre a tárban, bármelyiket betölthetjük a főprogram futása közben és azonnal végrehajthatjuk. Nagyméretű programokhoz írhatunk egy rövid kis menüprogramot, amely semmi egyebet nem csinál, mint egymás után betölti lemezről a szükséges programrészeket.

5.3 A grafika programozási elvei a STRUKTO 64-ben

A Commodore 64-esen grafikát készíteni nem túlságosan egyszerű feladat, hiszen a képernyő 8 × 8-as pontfelbontású karaktermezőiben kell a képernyő egyes pontjait megkeresni és programozni, ráadásul minden ilyen mezőn belül maximum két színt használhatunk. Lehetetlen egy mezőn belül pl. fekete háttérben két egymást keresztező piros és kék vonalat ábrázolni. A korlátozás oka, hogy a C 64-es minden ponthoz alapvetően két állapotot rendel; a pont látható vagy nem látható.
Ez a korlátozás sok esetben gondot okozhat, különösen akkor, ha a kirajzolandó vonalak egymáshoz viszonyított helyzete előre nem ismert. A STRUKTO 64 grafika programozási szemléletét a következőképpen lehet röviden összefoglalni:
A programot úgy készítjük, mintha a C 64-esen a színek tetszőleges kombinációit szabadon használhatnánk.
Tegyük fel, hogy egy 8 × 8-as mezőben két színt felhasználtunk: fekete háttérben húztunk egy kék vonalat balról jobbra, és most egy harmadik színnel felülről lefelé is akarunk egy vonalat húzni.
A harmadik szín megjelenítése csak úgy lehetséges, ha a kívánt pontokban már meglévő színek (fekete, kék) valamelyikét felcseréljük a harmadik színnel. A STRUKTO 64 mindig úgy választja meg a felcserélendő színt, hogy a harmadik szín megjelenítéséhez a lehető legkevesebb változtatásra legyen szükség.
A gyakorlatban a színes grafika előállítása nagyon egyszerű, ugyanis programozás közben nem kell tekintettel lennünk a Commodore 64-es sajátosságaira. Természetesen multicolor üzemmódban is programozhatunk, amelyben a felbontóképesség ugyan kisebb (félakkora), de pont emiatt négy szín áll rendelkezésünkre. Sőt ebben az esetben is élhetünk az előzőekben megismert lehetőségekkel, ha négy színnél is többre van szükségünk.

AZ ABLAK

A STRUKTO 64 grafikában definiálhatunk tetszőleges 8 × 8-as egységekből álló ún. ablakokat, amelyeket egyetlen színnel tölthetünk ki, és a képernyőn tetszőleges irányban (jobbra, balra, le, fel) görgethetünk (scroll funkció).
Azt is megtehetjük, hogy egy, az ablak egyik végét érintő sort vagy oszlopot megjelenítjük az ablak másik végén.
Az ablakok színét bármikor módosíthatjuk. A fenti lehetőségeket nagyon jól használhatjuk mozgó grafikák készítésére, vagy bizonyos adatok kiemelésére.

SZÖVEG A GRAFIKÁBAN

A STRUKTO 64 megengedi, hogy az ablakokban tetszőleges szöveget helyezzünk el úgy, hogy az ablakot az írás során külső egységként kezeljük, 17-es egységszámmal (ugyanúgy, ahogy a nyomtató száma pl. 4).
Ha írás közben kilépnénk az ablakból, a rendszer automatikusan egy görgetést hajt végre, így a grafikát tökéletesen úgy használhatjuk, mintha szövegképernyőn dolgoznánk. A grafikába dupla szélességű vagy magasságú betűket is írhatunk.
A kapott szövegeket tárolhatjuk lemezen.

A SPRITE-OK

A sprite-okat közvetlenül a képernyőről olvassa le a rendszer. Ez azt jelenti, hogyha parancs üzemmódban, PRINT utasításokkal felrajzoljuk a képernyőre az ábrát, a STRUKTO 64 az SHGET utasítás hatására leolvassa és tárolja a képernyőn látható táblázatot. Egyszerre 16 táblázatot tárolhatunk anélkül, hogy a tárolóterületről nekünk kellene gondoskodnunk.
Az SPDEF A,B,F... utasítással közölhetjük a géppel, hogy az a sorszámú sprite-ot, a b-ben megadott tárolási formában, f színben rajzolja ki a képernyő egy adott pozíciójára. A sprite-ot az SPPUT utasítással lehet a képernyőn mozgatni, az SPON/SPOFF utasításokkal pedig be-/kikapcsolni. A formátumtáblázat is tárolható.

MUSIK

A STRUKTO 64 három hangra komponált, szólamonként maximum 500 hangjegyből álló dallamokat képes tárolni és miközben a program fut, vagy a programozó dolgozik, megszólaltatni.
A hangjegyeket egyszerű formában közvetlenül begépelhetjük, pl. egy 1 másodpercig tartó (egész) C hang jelölése: "C64".
A hangjegyeket a rendszer egymástól elválasztva rakja be, úgy, hogy a tárban ne programterületet foglaljanak el.

EGY PÉLDAPROGRAM

Az elmondottakat a STRUKTO 64 rendszer alatt készített strukturált program bemutatásával szeretnénk szemléltetni. A feladatot a matematika területéről vettük: a program megkeresi az egy adott számnál kisebb prímszámokat az "eratoszthenészi rosta" alapelven. Az eljárás lényege a következő: a legkisebb prímszám a 2, a sorból tehát kihagyjuk az összes 2-vel osztható számot (2 többszöröseit), azaz a páros számokat. A következő legkisebb szám (a 3) már biztosan prímszám. Most kihagyjuk a sorból az ő összes többszöröseit. Az így megmaradt számok közül a legkisebb megint biztosan prímszám stb. A programnak az 1000-nél kisebb prímszámok meghatározásához kb. 28 másodpercre van szüksége, ami jóval kevesebb, mintha a keresést egyszerű osztással végeznénk.

```
P35

100 : REM A PRIMSZAMOK MEGHATAROZASA
110 : REM A ROSTA-ELV
120 : REM ALAPJAN:
130 : REM A MEGTALALT PRIMSZAM
140 : REM TOBBSZOROSET
150 : REM KIHAGYJUK.
160 : REM
170 : REM
180 : REPEAT
190 :    CLR
200 :    PRINT"";
210 :    CENTRE"PRIMSZAMOK"
220 :    PRINT
230 :    REPEAT
240 :       INPUT"2-TOL MEDDIG?";B
250 :    UNTIL B=INT(B) AND B<=15000 AND B>2
260 :    DIM A%(B/2)
```

```
270 :    PRINT "       2.";
280 :    FOR F=3 TO B STEP 2
290 :      REM
300 :      REM 3-TOL B-IG
310 :      REM KIHAGYJUK A PAROS SZAMOKAT
320 :      REM
330 :      IF A%(F/2)=0
340 :        REM
350 :        REM F PRIMSZAM
360 :        REM KIIRAS
370 :        REM
380 :        PRINTF;
390 :        I=3*F
400 :        REM
410 :        REM A TOBBSZOROS PARATLAN
420 :        REM SZAMOKAT KIHAGYJUK
430 :        REM
440 :        WHILE I<B
450 :          A%(I/2)=1
460 :             I=I+2*F
470 :        WHILEEND
480 :      IFEND
490 :    FOREND
500 :    PRINT
510 :    PRINT"TOVABB (I/N)";
520 :    REPEAT
530 :      GET A$
540 :    UNTIL A$="I" OR A$="N"
550 :  UNTIL A$="N"
560 :  END

READY.
```

– JEGYZET –

ÁRA: 308,-Ft

SZÁMÍTÁSTECHNIKA A KÖNYVESBOLTOKBAN

NOVOTRADE – 2C ÁRUHÁZ

1136 Bp., Balzac u. 35. Tel.: 402-954

Az alább felsorolt üzletekben már rendelkezésére állunk:

ÁLLAMI KÖNYVTERJESZTŐ V. - NOVOTRADE 2C

BUDAPEST

Táncsics Könyvesbolt
1073 Lenin krt. 17.
Telefon: 422-178

BUDAPEST

Műszaki Könyváruház
1061 Liszt Ferenc tér 9.
Telefon: 420-353

MŰVELT NÉP KÖNYVTERJESZTŐ V. - NOVOTRADE 2C

PÉCS

Zrínyi Miklós Könyvesbolt
7621 Jókai u. 25.
Telefon: 72-12835

SZOMBATHELY

Savaria Könyvesbolt
9700 Mártírok tere 1.
Telefon: 94-12341

SZEGED

Tömörkény Könyvesbolt
6720 Lenin krt. 48.
Telefon: 62-21453

VESZPRÉM

Kölcsey Ferenc Könyvesbolt
8200 Kossuth L. u. 8.

Minden érdeklődőt szeretettel vár az ÁKV, a
Művelt Nép és a NOVOTRADE RT.!